Model-based Reinforcement Learning

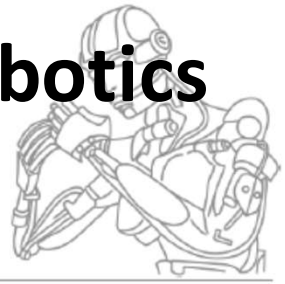
2018. 3.

AI Friends

김홍배, KARI

Policy Search Methods for Robotics

Gerhard Neumann, University of Lincoln
Jan Peters, TU Darmstadt



Outline

Taxonomy of Policy Search Algorithms

Model-Free Policy Search Methods

- Policy Gradients
- Natural Gradients
- Exact Information Geometric Updates
- Success Matching

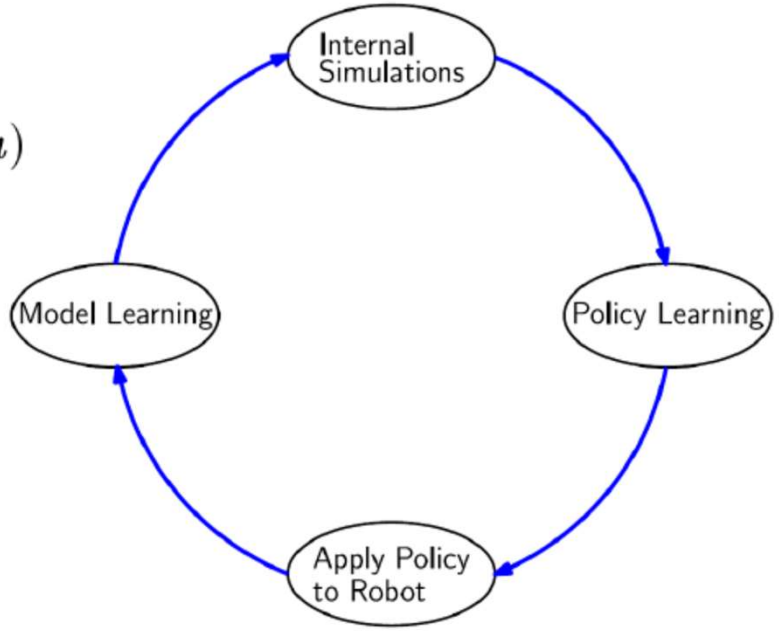
Model-Based Policy Search Methods

- Greedy Updates: PILCO [Deisenroth & Rasmussen, 2011]
- Bounded Updates: Model-Based REPS [Peters at al., 2010], Guided Policy Search by Trajectory Optimization [Levine & Koltun, 2014]

Model-based Policy Search Methods

Learn dynamics model from data-set

$$\mathcal{D} = \left\{ \left(\boldsymbol{s}_{1:T}^{[i]}, \boldsymbol{a}_{1:T-1}^{[i]} \right) \right\} \rightarrow \hat{p}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) \approx p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$$



Use learned model as simulator

- Sampling [Kupcsik, Diesenroth, Peters & Neumann 2013][Ng 2000]
- (Approximate) probabilistic Inference [Deisenroth & Rasmussen 2011, Levine & Koltun, 2014]

Update Policy

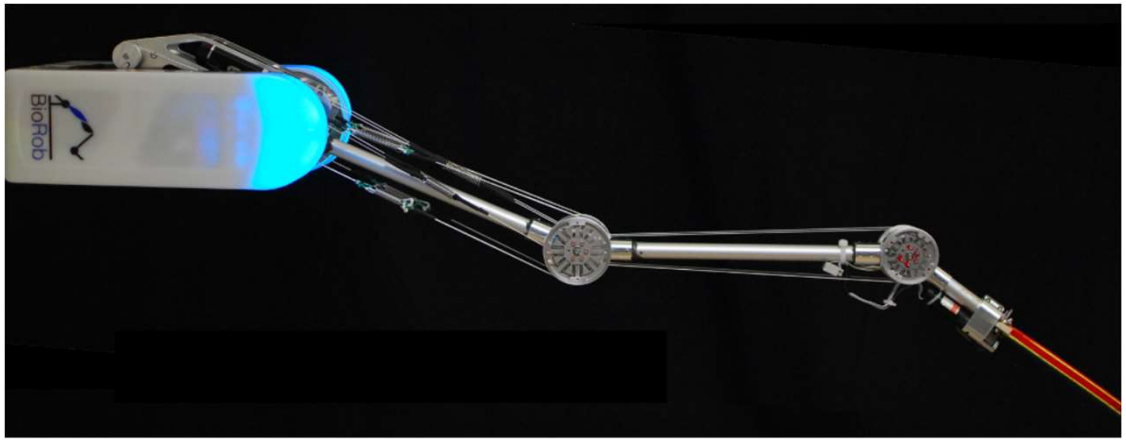
- Model-free methods on virtual sample trajectories [Kupcsik, Diesenroth, Peters & Neumann 2013]
- Analytic Policy Gradients [Deisenroth & Rasmussen, 2011]
- Trajectory optimization [Levine & Koltun, 2014]

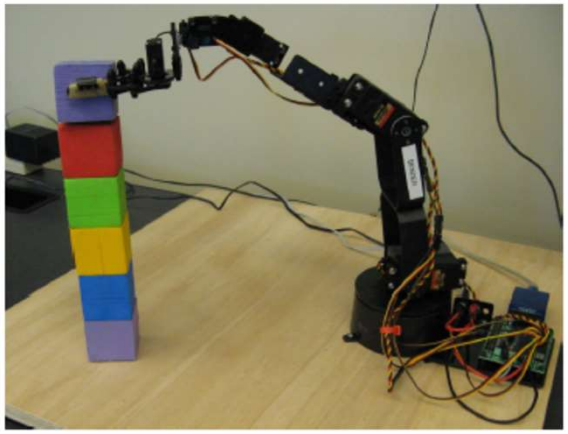
Autonomous Planning and Control

with Bayesian Nonparametric Models

PILCO: A Model-Based and Data-Efficient Approach to Policy Search

Model Learning in Robotics/Control





- Models in robotics are important for planning and control
- Typically: rigid-body dynamics, parameter identification
- Robots violate these assumptions quite often
- Nonparametric models can be very useful here :
  - Flexible model for data-driven extraction of relevant information
  - Make assumption at more intuitive level (e.g., smoothness instead of friction parameters)

Problem Formulation

Objective

Learn a **policy** π^* that yields minimal **expected long-term cost** $J^\pi(\boldsymbol{\theta})$

$$J^\pi(\boldsymbol{\theta}) = \sum_{t=0}^{T} \mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)|\pi]$$

Follow π for T steps starting from $p(\mathbf{x}_0)$

- **Policy parameters** $\boldsymbol{\theta}$
- **Cost** $c(\mathbf{x}_t)$ of being in state $\mathbf{x}_t$. We choose

$$c(\mathbf{x}_t) = 1 - \exp(-\tfrac{1}{2}\|\mathbf{x}_t - \mathbf{x}_{\text{target}}\|^2/\sigma_c^2) \quad \in [0, 1]$$

Challenges:

- **Data-efficient** solution (few trials)
- **Unknown transition dynamics** $f : (\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \mapsto \mathbf{x}_t$
- **No expert knowledge/demonstrations** available ➞ learn from scratch

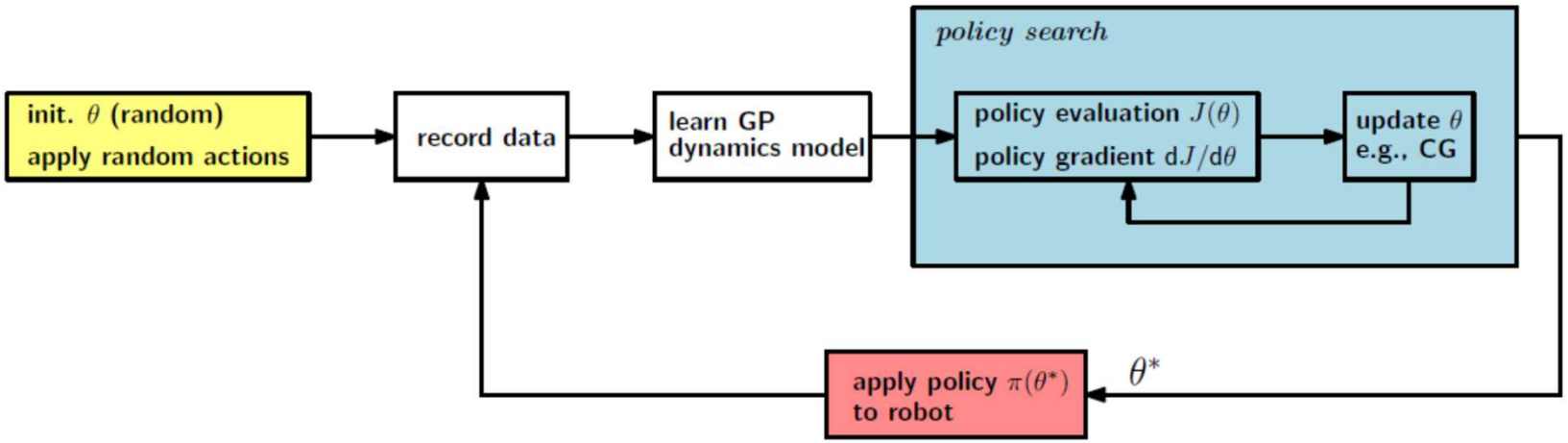
Making RL Efficient

Model-based RL

- Learn model of transition dynamics f
- Use model for internal simulation ➞ certainty equivalence assumption (Schneider, NIPS 1997; Bagnell and Schneider, ICRA 2001)
- Learn policy based on these simulations
- Hope: few interactions with system
  ➞ suffers from **model errors**, but can be **data efficient**

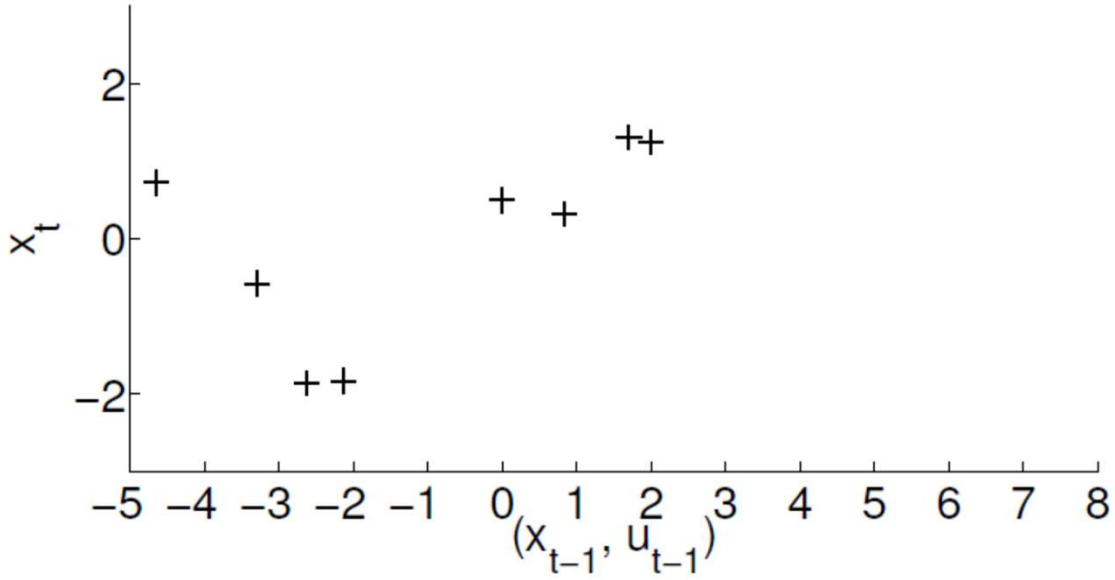
PILCO

- Probabilistic inference for learning control
- Model-based **policy search** method with **analytic policy gradients**
  ➞ find good policy parameters $\boldsymbol{\theta}^*$
- **Gaussian processes** for probabilistic dynamics model
  zero prior mean, SE covariance function
- Explicitly describe **model uncertainties**
  ➞ Take them into account during planning
  ➞ Reduce effect of model errors
  ➞ Allows for learning from scratch (episodic tasks)



Learning Nonparametric Transition Models

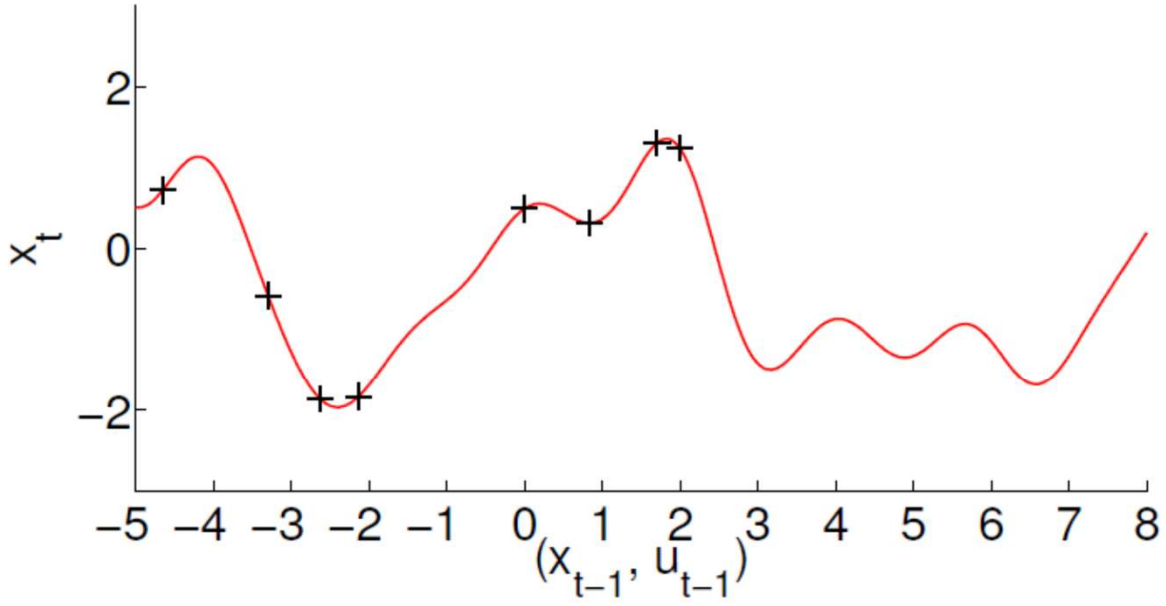
Learning a forward model: Find a transition function
$f : (x_{t-1}, u_{t-1}) \mapsto x_t$



Observed function values

Learning Nonparametric Transition Models

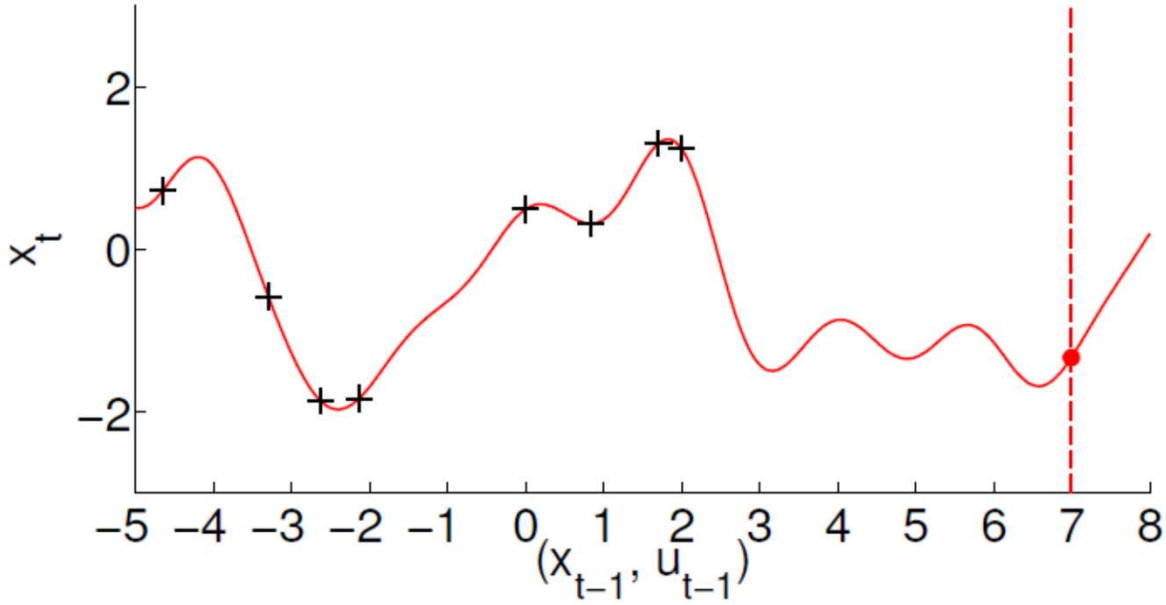
Learning a forward model: Find a transition function
$f : (x_{t-1}, u_{t-1}) \mapsto x_t$



Plausible function approximator

Learning Nonparametric Transition Models

Learning a forward model: Find a transition function
$f : (x_{t-1}, u_{t-1}) \mapsto x_t$

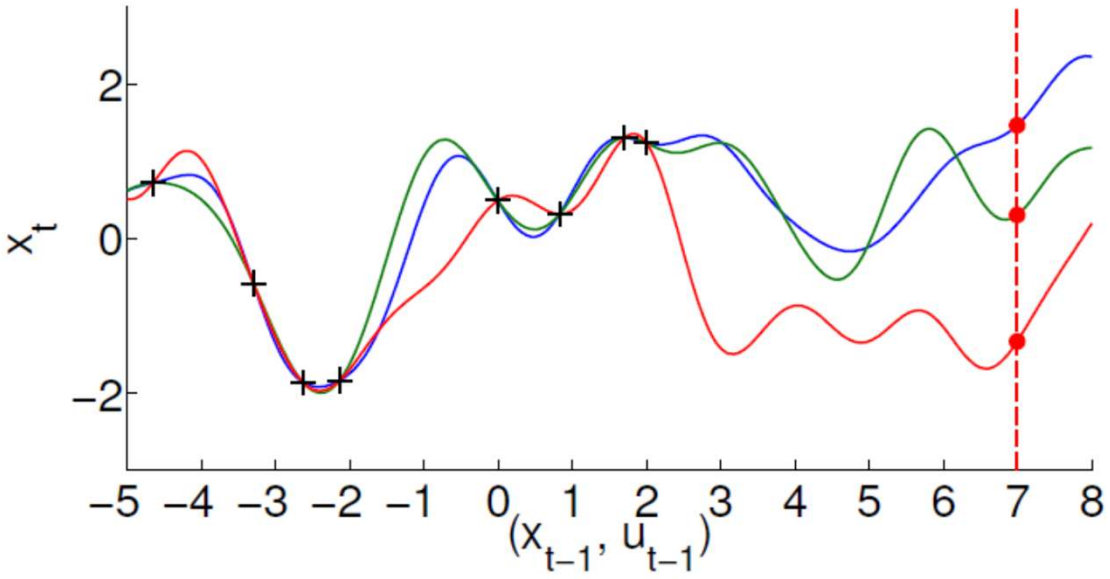


Plausible function approximator

Predictions? Decision Making?

Learning Nonparametric Transition Models

Learning a forward model: Find a transition function
$f : (x_{t-1}, u_{t-1}) \mapsto x_t$

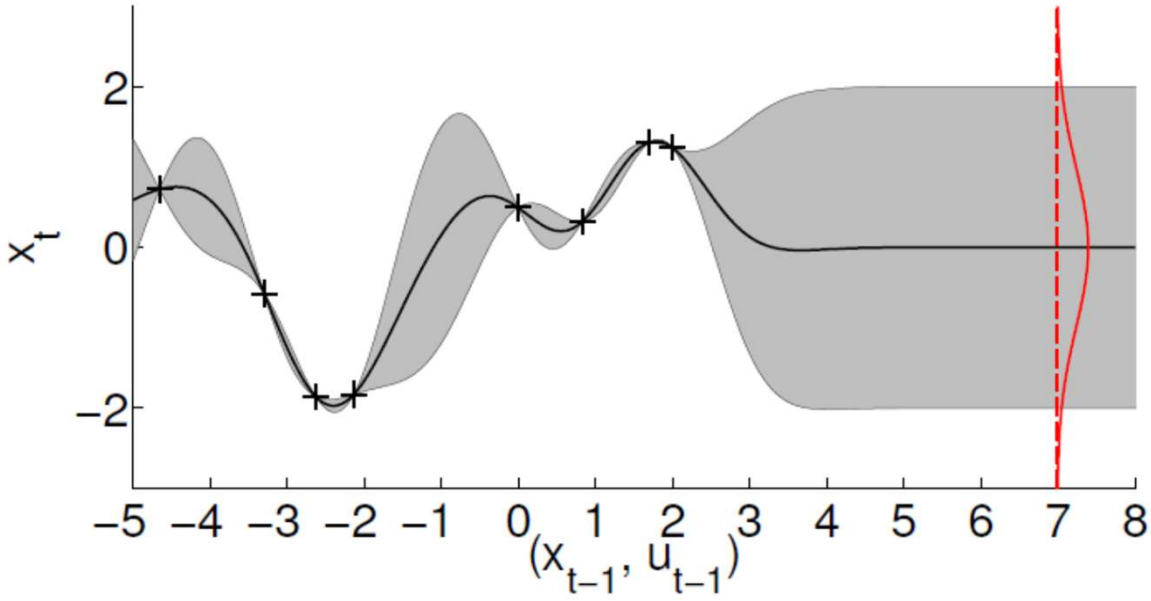


Plausible function approximator

Predictions? Decision Making? Model Errors!

Learning Nonparametric Transition Models

Learning a forward model: Find a transition function
$f : (x_{t-1}, u_{t-1}) \mapsto x_t$



Distribution over plausible functions

- ✓ Express uncertainty about the underlying function
- ✓ Probabilistic models

Controller Learning

Objective (Control Learning/Policy Search)

Find policy parameters θ^* that minimize the expected long-term cost

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}[c(x_t)|\theta]\,, \qquad p(x_0) = \mathcal{N}(\mu_0, \Sigma_0)\,.$$

- Markovian transition dynamics
  $x_{t+1} = f(x_t, u_t) + w$
- Controls $u_t = \pi(x_t, \theta)$
- Policy π
- Policy parameters θ
- Cost function c, e.g., $\|x - x_{\text{target}}\|^2$

Model-based Policy Search

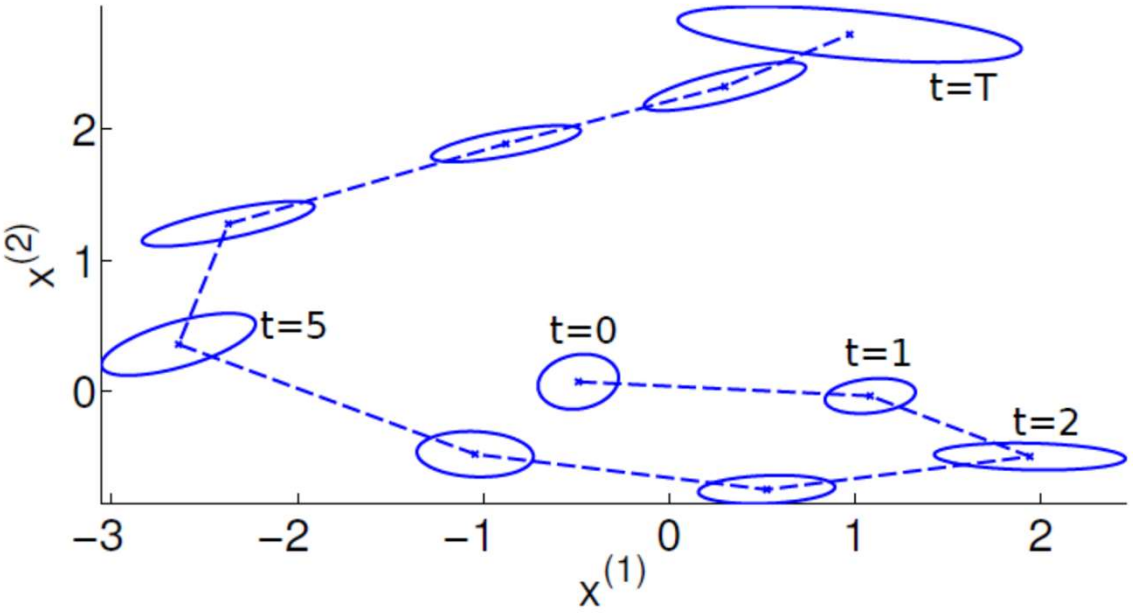
Objective:

Minimize expected long-term cost $J(\theta) = \sum_t \mathbb{E}[c(x_t)|\theta]$

Algorithm:

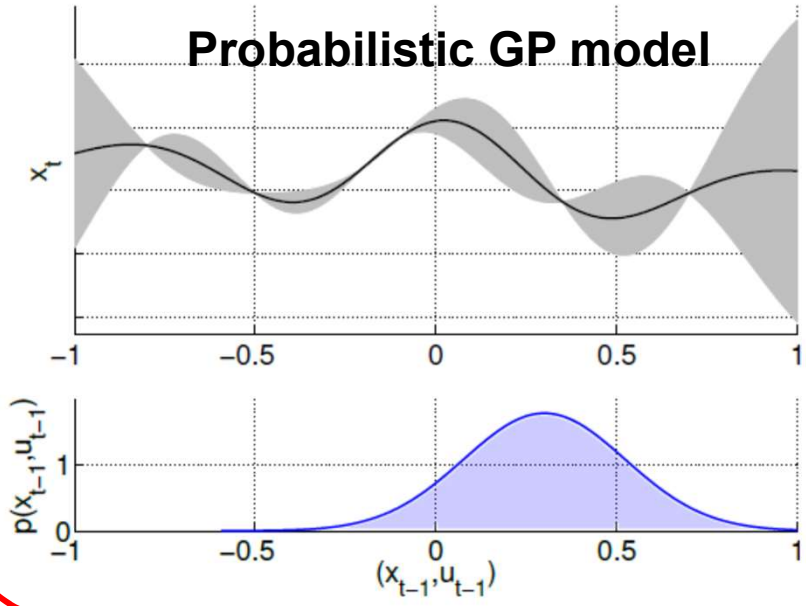
1. Probabilistic GP model for transition function f to be robust to model errors
2. Compute long-term predictions $p(x_1)$, $p(x_2)$, …… $p(x_T)$
3. Policy Learning
   - ✓ Compute expected long-term cost $J(\theta)$
   - ✓ Find parameters θ that minimize $J(\theta)$

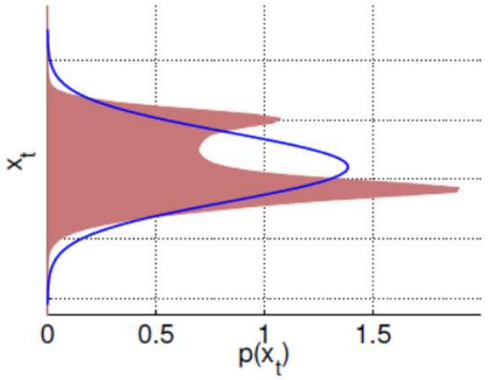
Long-Term Predictions



- Iteratively compute $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$

Long-Term Predictions





Moment Matching

- Iteratively compute $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$

$$p(\boldsymbol{x}_t) = \iiint \underbrace{p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1},\boldsymbol{u}_{t-1})}_{\text{GP pred. distribution}} \underbrace{p(\boldsymbol{x}_{t-1},\boldsymbol{u}_{t-1})}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})} \mathrm{d}f\,\mathrm{d}\boldsymbol{x}_{t-1}\mathrm{d}\boldsymbol{u}_{t-1}$$

➔ Approximate inference : determine a centroid & variance

✓ Moment matching (Quinonero-Candela et al., 2003)

Moment Matching

deterministic approximate inference

$p(x_t)$ can now be approximated with $\mathcal{N}(\mu_t, \Sigma_t)$ where

$$\mu_t = \mu_{t-1} + \mu_\Delta$$
$$\mathbf{\Sigma}_t = \mathbf{\Sigma}_{t-1} + \mathbf{\Sigma}_\Delta + \mathrm{cov}[\mathbf{x}_{t-1}, \Delta_t] + \mathrm{cov}[\Delta_t, \mathbf{x}_{t-1}]$$
$$\mathrm{cov}[\mathbf{x}_{t-1}, \Delta_t] = \mathrm{cov}[\mathbf{x}_{t-1}, \mathbf{u}_{t-1}]\mathbf{\Sigma}_u^{-1}\mathrm{cov}[\mathbf{u}_{t-1}, \Delta_t]$$

μ_Δ and Σ_Δ are computed exactly via iterated expectation and variance

$$\mu_\Delta^a = \mathbb{E}_{\tilde{\mathbf{x}}_{t-1}}[\mathbb{E}_f[f(\tilde{\mathbf{x}}_{t-1})|\tilde{\mathbf{x}}_{t-1}]]$$

$$\sigma_{aa}^2 = \mathbb{E}_{\tilde{\mathbf{x}}_{t-1}}\left[\mathrm{var}_f[\Delta_a|\tilde{\mathbf{x}}_{t-1}]\right] + \mathbb{E}_{f,\tilde{\mathbf{x}}_{t-1}}[\Delta_a^2] - (\mu_\Delta^a)^2$$
$$\sigma_{ab}^2 = \mathbb{E}_{f,\tilde{\mathbf{x}}_{t-1}}[\Delta_a \Delta_b] - \mu_\Delta^a \mu_\Delta^b\,, \quad a \neq b\,,$$

Model-based Policy Search

Objective

Minimize expected long-term cost $J(\theta) = \sum_t \mathbb{E}[c(x_t)|\theta]$

High-Level Steps:

1. Probabilistic model for transition function f to be robust to model errors
2. Compute long-term predictions $p(x_1|\theta), \ldots, p(x_T|\theta)$
3. **Policy improvement**
   - Compute expected long-term cost $J(\theta)$
   - Find parameters θ that minimize $J(\theta)$
4. Apply controller

Policy Improvement

Objective

Minimize expected long-term cost $J(\boldsymbol{\theta}) = \sum_t \mathbb{E}[c(\boldsymbol{x}_t)|\boldsymbol{\theta}]$

- Know how to predict $p(\boldsymbol{x}_1|\boldsymbol{\theta}), \ldots, p(\boldsymbol{x}_T|\boldsymbol{\theta})$
- Compute

$$\mathbb{E}[c(\boldsymbol{x}_t)|\boldsymbol{\theta}] = \int c(\boldsymbol{x}_t)\mathcal{N}\left(\boldsymbol{x}_t \,|\, \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\right) \mathrm{d}\boldsymbol{x}_t\,, \quad t = 1, \ldots, T\,,$$

  and sum them up to obtain $J(\boldsymbol{\theta})$

- Analytically compute gradient $\mathrm{d}J(\boldsymbol{\theta})/\mathrm{d}\boldsymbol{\theta}$
- Standard gradient-based optimizer (e.g., BFGS) to find $\boldsymbol{\theta}^*$

Guided Policy Search

Deep Robotics

J.hyeon Park

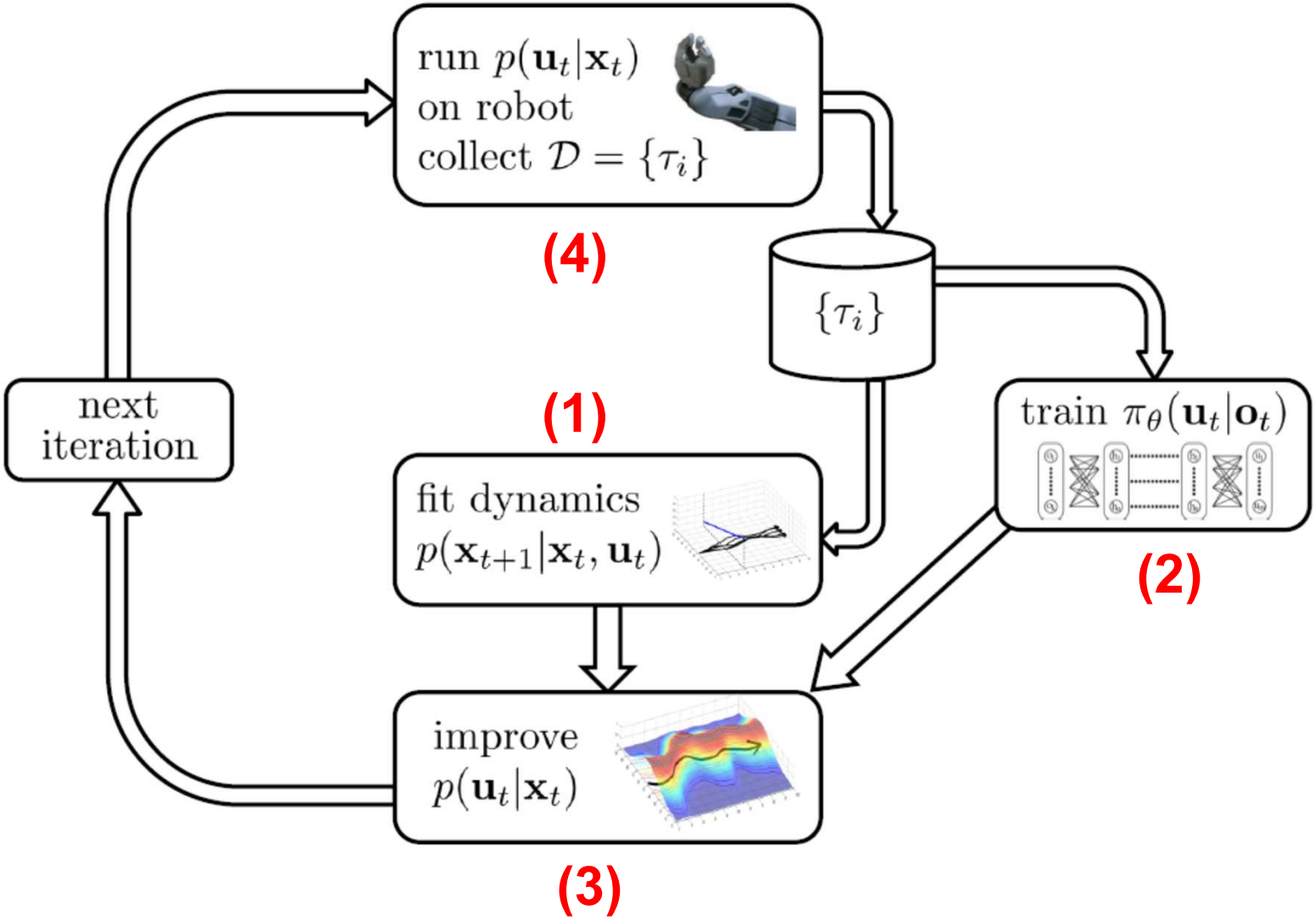
Guided Policy Search

Sergey Levine

Guided policy search

- Levine, Sergey, and Vladlen Koltun. "Guided policy search."*Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 2013.
- Sergey Levine, Pieter Abbeel. *Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics*. NIPS 2014.
- William Montgomery, Sergey Levine. *Guided Policy Search as Approximate Mirror Descent*. NIPS 2016.

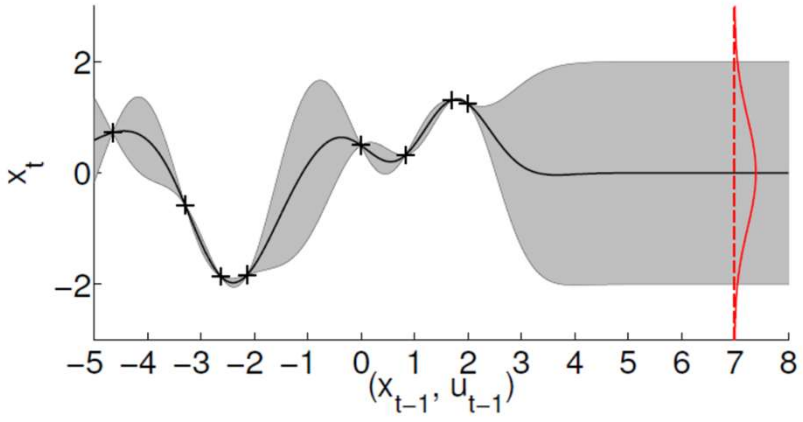
Work flow of Guided Policy Search



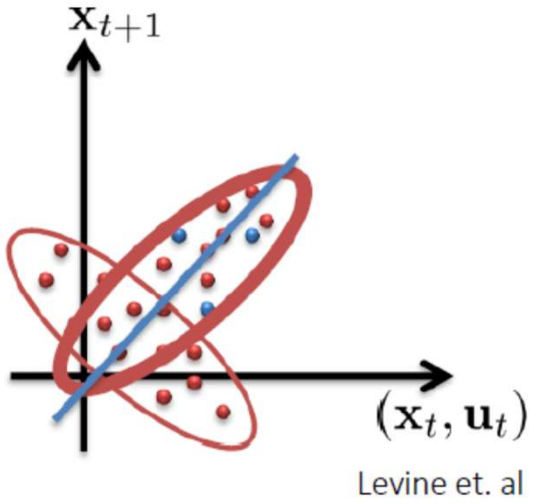
1. Learning Dynamics(fit dynamics, state estimator)

$$Get\ P(x_{t+1}|x_t, u_t)$$

- Global Model



Gaussian Processing



Gaussian Mixture Model

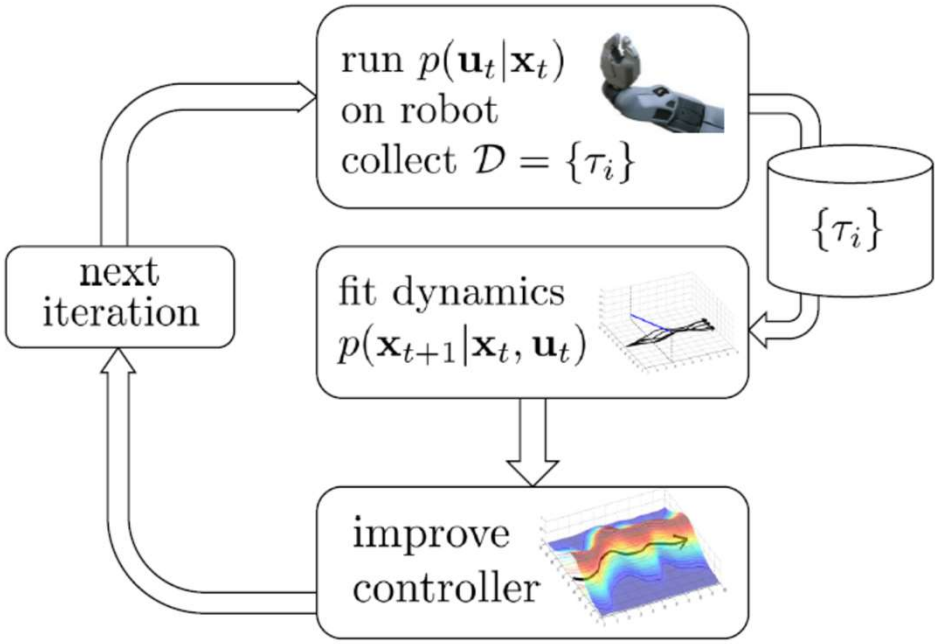
1. Learning Dynamics(fit dynamics, state estimator)

- Local Model : Time varying linear model

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t), \Sigma)$$

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t$$

$$\mathbf{A}_t = \frac{df}{d\mathbf{x}_t} \qquad \mathbf{B}_t = \frac{df}{d\mathbf{u}_t}$$



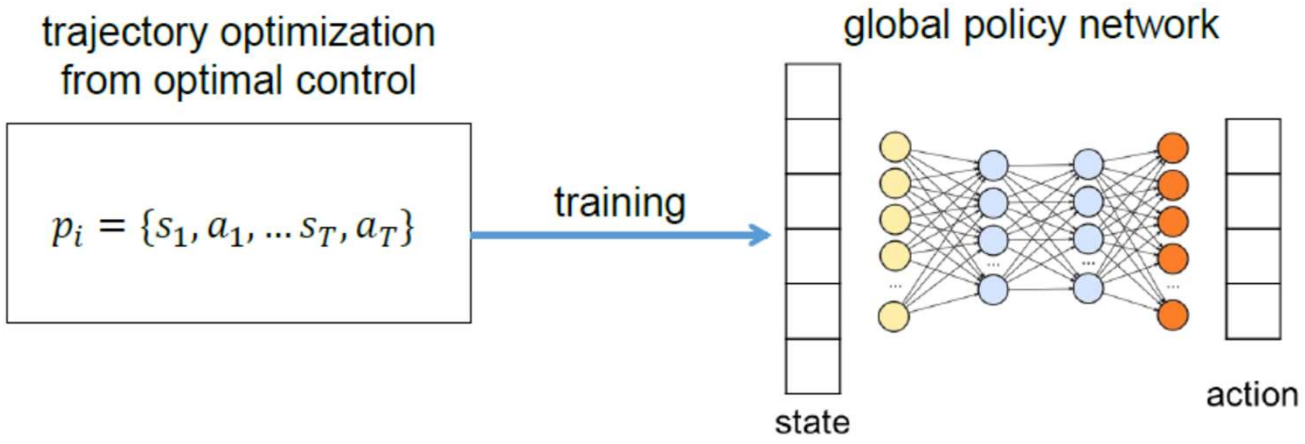
idea: just fit $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}$ around current trajectory or policy!

Bayesian Linear dynamics fitting

2. Train Policy Networks, π_θ

Supervised learning

$$\pi_\theta \leftarrow argmin_\theta \sum KL(\pi_\theta || p_i)$$



3. Trajectory Optimization

Get $P(u_t|x_t)$

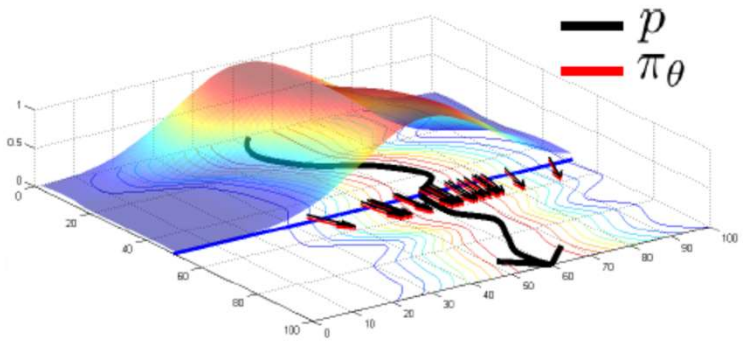
Trajectory optimization with constraints, $KL(p_i||\pi_\theta)$

Prior information of Policy

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

$$\text{s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

$\pi_\theta(x_t)$: *Policy Networks trained in step 2*



4. Operate real plants or Robotics

Run $P(u_t|x_t)$

and collect data $\{x_t, u_t, r_t, x_{t+1}\}$

Bayesian Linear dynamics fitting

Fit a *Global* Gaussian Mixture Model using all samples (x_t, u_t, x_{t+1}) of all iterations and time steps. -> prior

Use current samples (from this iteration) and obtain Gaussian posterior for (x_t, u_t, x_{t+1}), which you condition to obtain $p(x_{t+1}|x_t, u_t)$.

Such prior results in 4 to 8 times less samples needed, despite the fact that it is not accurate enough by itself.

$$\Sigma = \frac{\mathbf{\Phi} + N\hat{\Sigma} + \frac{Nm}{N+m}(\hat{\mu} - \mu_0)(\hat{\mu} - \mu_0)^{\mathrm{T}}}{N + n_0}$$

$$\mu = \frac{m\mu_0 + n_0\hat{\mu}}{m + n_0}.$$

Posterior of mean and covariance where $\hat{\mu}, \hat{\Sigma}$ are the empirical means and covariances and Φ, μ_0, n_0, m an inverse Wishart prior

Learning Neural Network Policies with guided Policy Search under Unknown Dynamics, Levine and Abbeel 2014

Bayesian Linear dynamics fitting

Fit a *Global* Model of Dynamics by fitting a Neural Network using all samples (x_t, u_t, x_{t+1}) of all iterations and time steps, and across multiple manipulation tasks->multi-task learning.

Use model predictive control with iLQR for computing the policy at every time step.

State is the robotic arm configuration and cost depends on a desired end-effector pose. No object involved in the state.

$$\bar{f}([\mathbf{x};\mathbf{u}]) \approx \bar{f}([\mathbf{x}_i;\mathbf{u}_i]) + \frac{d\bar{f}}{d[\mathbf{x};\mathbf{u}]}^{\mathrm{T}} ([\mathbf{x};\mathbf{u}] - [\mathbf{x}_i;\mathbf{u}_i])$$

$$\bar{\mu} = \begin{bmatrix} [\mathbf{x}_i;\mathbf{u}_i] \\ \bar{f}([\mathbf{x}_i;\mathbf{u}_i]) \end{bmatrix}$$

$$\bar{\Sigma} = \begin{bmatrix} \bar{\Sigma}_{\mathbf{xu},\mathbf{xu}} & \bar{\Sigma}_{\mathbf{xu},\mathbf{xu}} \frac{d\bar{f}}{d[\mathbf{x};\mathbf{u}]} \\ \frac{d\bar{f}}{d[\mathbf{x};\mathbf{u}]}^{\mathrm{T}} \bar{\Sigma}_{\mathbf{xu},\mathbf{xu}} & \frac{d\bar{f}}{d[\mathbf{x};\mathbf{u}]}^{\mathrm{T}} \bar{\Sigma}_{\mathbf{xu},\mathbf{xu}} \frac{d\bar{f}}{d[\mathbf{x};\mathbf{u}]} + \bar{\Sigma}_{\mathbf{x}',\mathbf{x}'} \end{bmatrix}$$

One shot Learning of Manipulation Skills with Online Dynamic Adaptation and Neural Network Priors, Fu et al

Probabilistic Trajectory Matching

Key Idea:

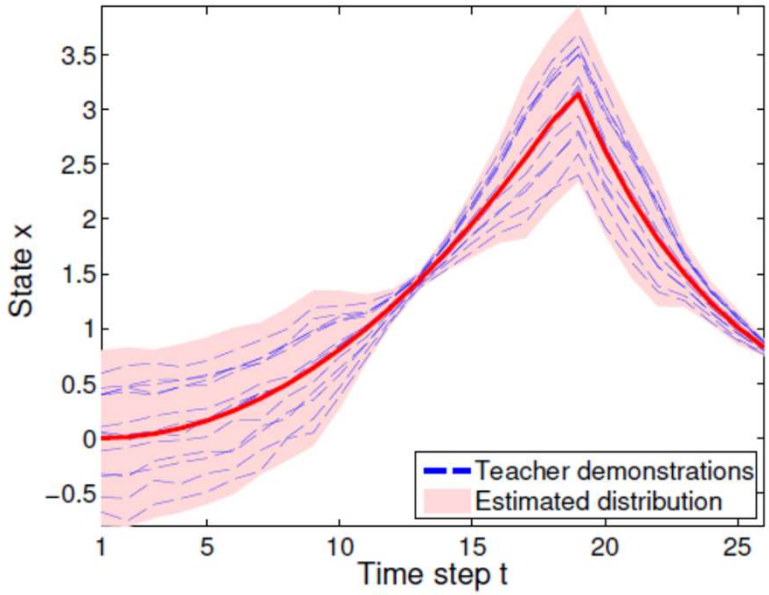
- Directly match the robot's predicted state trajectory $\boldsymbol{\tau}^{\pi}$ with the demonstrated trajectories $\boldsymbol{\tau}^{\text{demo}}$
- Account for uncertainty
  - Noisy demonstrations
    - ⏩ Distribution $p(\boldsymbol{\tau}^{\text{demo}})$ over demonstrated trajectory
  - Uncertain dynamics model
    - ⏩ Distribution $p(\boldsymbol{\tau}^{\pi})$ over predicted trajectory

Objective: Probabilistic Trajectory Matching

$$\pi^* \in \arg\min_{\pi} \mathrm{KL}\big(p(\boldsymbol{\tau}^{\text{demo}}) || p(\boldsymbol{\tau}^{\pi})\big)$$

Imitation Learning & Guide Policy Search

Trajectory Representation



- State trajectory $\boldsymbol{\tau} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$
- Factorizing assumption: $p(\boldsymbol{\tau}) \approx \prod_t p(\boldsymbol{x}_t)$
- Gaussian approximation of the marginals $p(\boldsymbol{x}_t) \approx \mathcal{N}\left(\boldsymbol{x}_t \,|\, \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\right)$

Induced Cost Function

$$J_{\mathrm{IL}} = \mathrm{KL}\left(p(\boldsymbol{\tau}^{\mathrm{demo}}) || p(\boldsymbol{\tau}^{\pi})\right) \approx \sum_{t=1}^{T} \mathrm{KL}\left(p(\boldsymbol{x}_t^{\mathrm{demo}}) || p(\boldsymbol{x}_t^{\pi})\right) = \sum_{t=1}^{T} c(\boldsymbol{x}_t^{\pi}) = J_{\mathrm{RL}}$$

- Probabilistic trajectory matching induces a "natural" RL cost function c
- RL algorithms for solving imitation learning when we use $\mathrm{KL}\left(p(\boldsymbol{x}_t^{\mathrm{demo}}) || p(\boldsymbol{x}_t^{\pi})\right)$ as immediate cost function c

What is optimal control ?

$$\min_{a_1, a_2, \ldots a_T} \sum L(s_t, a_t)$$

Cost rate

under dynamics $s_{t+1} = f(s_t, a_t)$

optimal control example : iLQR

Assumption :

dynamics is linear $s_{t+1} = F_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + \mathrm{f_t}$

cost is quadratic $L(s_t, a_t) = \frac{1}{2} \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T C_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T c_t$

optimal control example : iLQR

Dynamic Programming !

At time T

Q –function : $Q(s_T, a_T) = cost + \frac{1}{2}\begin{bmatrix} s_T \\ a_T \end{bmatrix}^T C_T \begin{bmatrix} s_T \\ a_T \end{bmatrix} + \begin{bmatrix} s_T \\ a_T \end{bmatrix}^T c_T$

~ Cost rate
@ a given time

Given terminal cost

derivative of Q : $\nabla_{a_T} Q(s_T, u_T) = C_{a_T, s_T} s_T + C_{a_T, a_T} a_T + c_{a_T}^T$ **= 0**

optimal action : $a_T = -C_{a_T, a_T}^{-1}\left(C_{a_T, s_T} s_T + c_{a_T}\right)$

optimal control example : iLQR

At time T

with dynamics : $s_T = F_T \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix} + \mathrm{f_T}$

$$a_T = -C_{a_T,a_T}^{-1} \left(C_{a_T,s_T} \left(F_T \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix} + \mathrm{f_T} \right) + c_{a_T} \right)$$

optimal control example : iLQR

At time $T-1$

Q –function: $Q(s_{T-1}, a_{T-1}) = Q(s_T, a_T) + \frac{1}{2}\begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix}^T C_{T-1} \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix} + \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix}^T c_{T-1}$

quadratic function of s_T, a_T

derivative of Q : $\nabla_{a_{T-1}} Q(s_{T-1}, u_{T-1}) = \ \ldots$ **= 0**

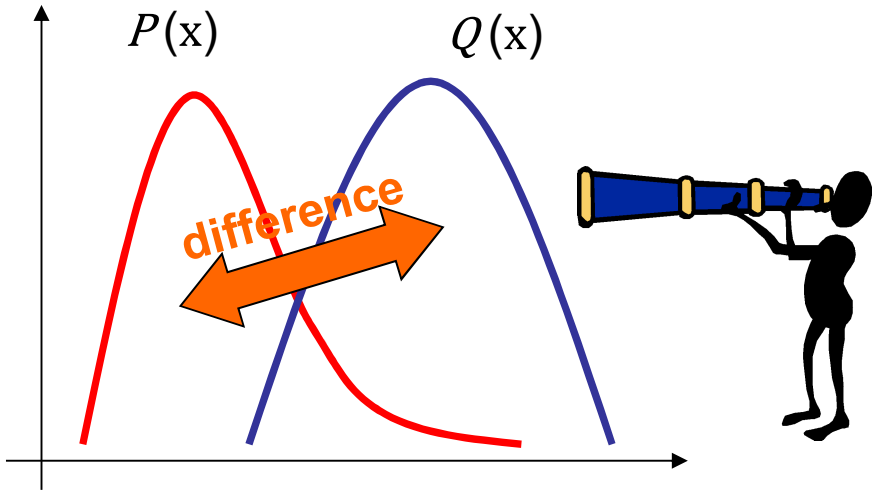
optimal action : $a_{T-1} = \ldots$

➔ We can get pairs of state & control from start to terminal

Kullback-Leibler (KL) divergence

- A measure of the difference between two probability distributions: $P(\mathrm{x})$ and $Q(\mathrm{x})$

$$D(P\|Q) \equiv \int P(\mathrm{x}) \log \frac{Q(\mathrm{x})}{P(\mathrm{x})}\, dx$$



We can measure the difference according to an objective and numerical value.

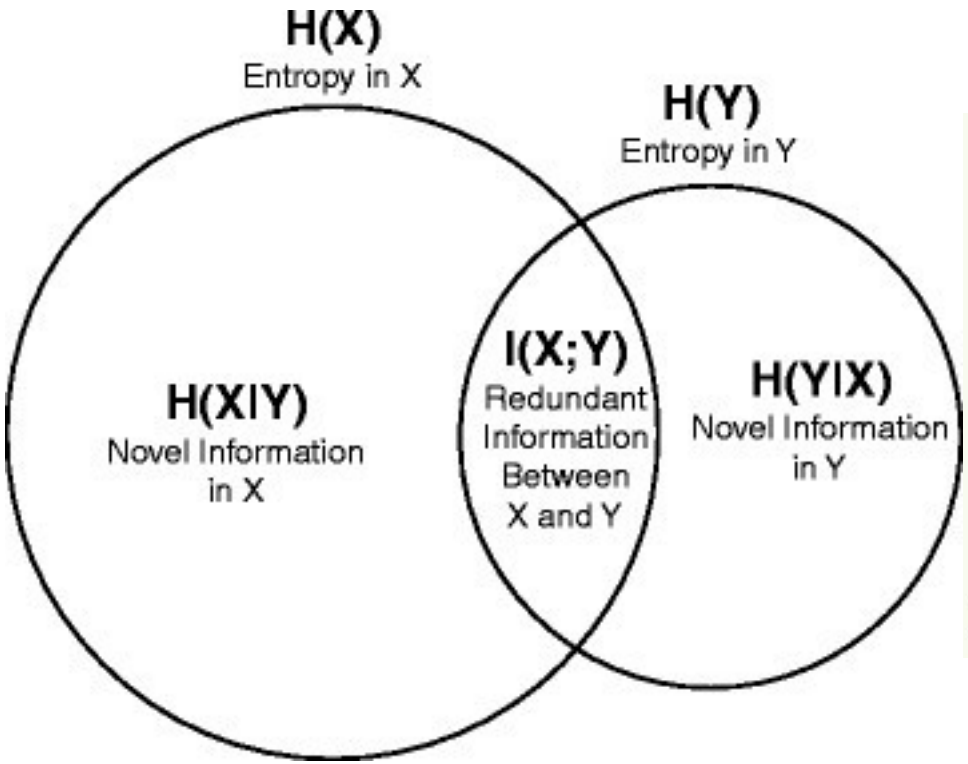
Note: KL divergence is not a metric.

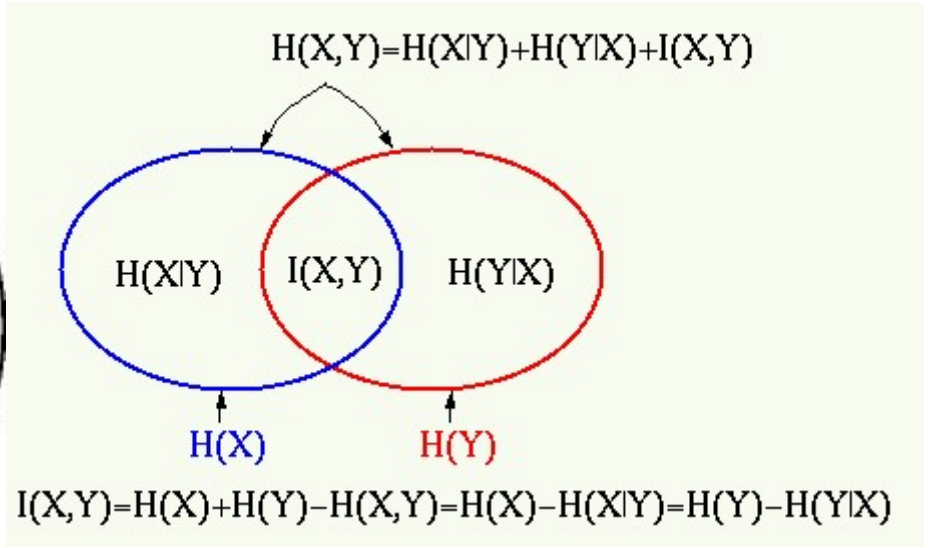
$$D(P\|Q) \neq D(Q\|P)$$

Joint Entropy

Venn diagram for definition of entropies

Conditional Entropy H(X|Y)





Relation among Entropies

$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$$

$$H(X|Y) = H(X,Y) - H(X)$$

KL Divergence

$$H(X|Y) = -\sum_{i=1}^{m}\sum_{j=1}^{n} p(x_i, y_j)\cdot \log p(x_i|y_j)$$

Bay's theorem

$$= \sum_{i=1}^{m}\sum_{j=1}^{n} p(x_i, y_j)\cdot \log \frac{p(y_j)}{p(x_i,y_j)}$$

Kullback-Leibler (KL) divergence

KL Divergence

$$K(P||Q) = -\sum_i log\left(\frac{q_i}{p_i}\right)p_i$$

-. P = Target Distribution (True Distribution)

If distribution *p* is equal to *q*, KL divergence is 0

$$if\ p = q, K(P||Q) = -\sum_i log\left(\frac{q_i}{p_i}\right)p_i = 0\ because\ p_i = q_i\ and\ log(1) = 0$$

$$if\ p \neq q, K(P||Q) = -\sum_i log\left(\frac{q_i}{p_i}\right)p_i \neq 0$$

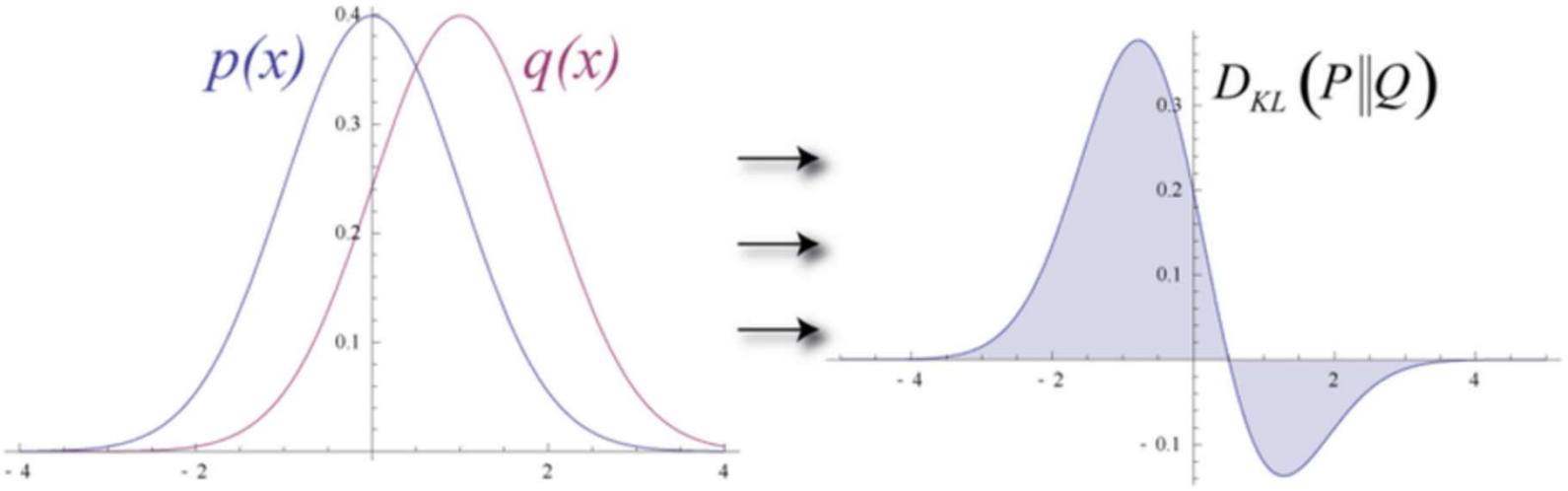
$\frac{q_i}{p_i}$ **could be thought as a statistic**

$$K(P||Q) = -\sum_i log\left(\frac{q_i}{p_i}\right)p_i$$

$\frac{q_i}{p_i}$ is a statistic of H0: $p = q$ *vs* H1: p ≠ q

Kullback-Leibler (KL) divergence

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}$$

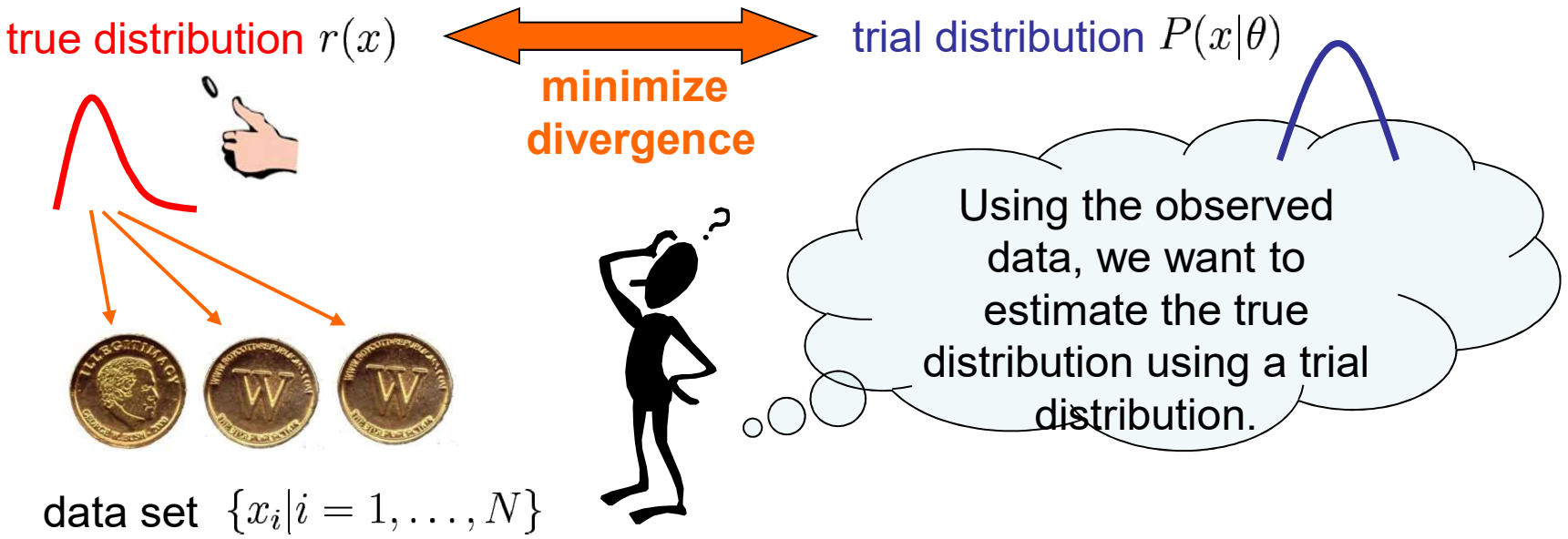


$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) &= -\sum_x p(x) \log q(x) + \sum_x p(x) \log p(x) \\ &= H(P,Q) - H(P) \end{aligned}$$

KL divergence = Conditional Entropy H(P|Q)

Minimize KL divergence

- Random events are drawn from the real distribution



The smaller the KL divergence $D(r||P)$, the better an estimate.

Minimize KL divergence

- KL divergence between the two distributions

$$D(r||P) = \int dx r(x) \log\left(\frac{r(x)}{P(x|\theta)}\right)$$

$$= \underbrace{\int dx r(x) \log r(x)}_{} - \underline{\int dx r(x) \log P(x|\theta)}$$

Constant: independent of parameter θ

To minimize KL divergence, we have only to maximize the second term with respect to the parameter θ.

Likelihood and KL divergence

- The second term is approximated by the sample mean:

$$\int dx r(x) \log P(x|\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \log P(x_i|\theta)$$

data set $\{x_i | i = 1, \ldots, N\}$

Log likelihood

$$l(\theta) = \log L(\theta) = \sum_{i=1}^{N} \log P(x_i|\theta)$$

They are the same:
- Minimizing the KL divergence
- Maximizing the likelihood