Data efficient reinforcement learning
Gaussian process, PILCO and DeepPILCO

Ali Younes

Bauman Moscow State Technical University

ay20-5-1994@hotmail.com

March 6, 2019

Outline

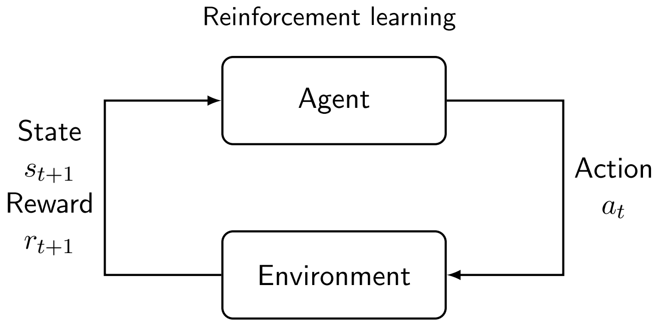
Introduction
- Data efficient reinforcement learning
- Gaussian Processes in RL

PILCO
- PILCO Algorithm
- PILCO Algorithm components
- Experimental results
- Experimental results

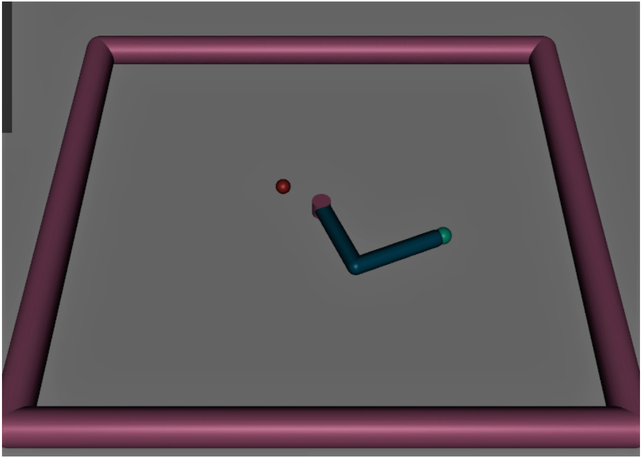
Deep PILCO
- PILCO problems

Why data efficient reinforcement learning?

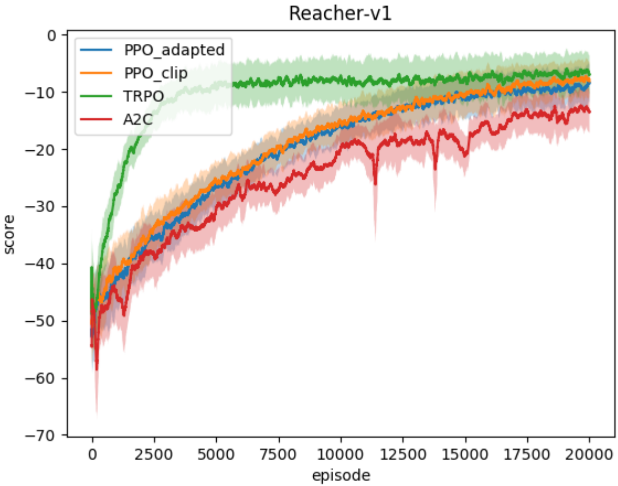
- Reinforcement learning is a hot research topic nowadays.
  It is considered one of the major research directions in robot learning.
- We have worked with model free algorithms (TRPO, PPO).



Why data efficient reinforcement learning?



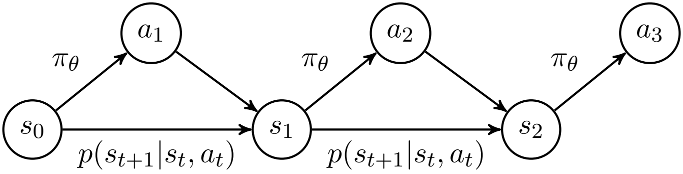
Why data efficient reinforcement learning?



Why data efficient reinforcement learning?

- In real world that means weeks of training.
  Collecting samples is expensive in real world robots
- We need a robot learning framework with as less as possible interaction time in the real world.
  Two solution:
  1. Transfer learning from simulation to real world (Sim2Real)
  2. Data efficient learning

Model Based reinforcement learning

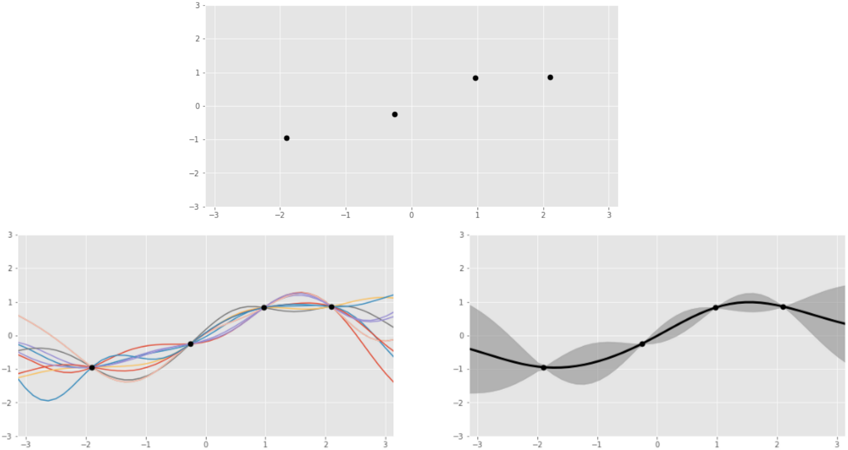


- We will use the following terminology:
  State : x_t , Control (action): u_t , Cost (reward): c
- Transition function: $x_{t+1} = f(x, u_t) + \omega$
- Control: $u_t = \pi(x_t, \theta)$
- The expected long-term cost: $J(\theta) = \sum_{t=1}^{T} \mathbb{E}[c(x_t)|\theta]$

Gaussian processes

- In probability theory and statistics, a Gaussian process is a stochastic process (a collection of random variables indexed by time or space), such that every finite collection of those random variables has a multivariate normal distribution, i.e. every finite linear combination of them is normally distributed.
- In other words, a Gaussian process is a probability distribution over possible functions.
- Gaussian process defined by:
    1. Mean function $m(.)$
    2. Covariance function (Kernel) $k(.,.)$

Gaussian processes - Regression



PILCO Algorithm

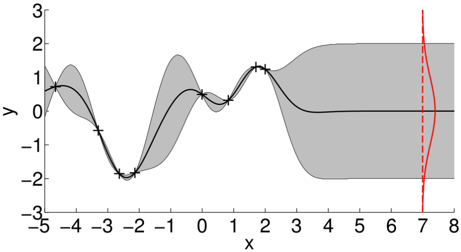
- M. Deisenroth and C. Rasmussen,
  "PILCO: A model-based and data-efficient approach to policy search",ICML-2011
  M. Deisenroth, D. Fox, and C. Rasmussen,
  "Gaussian processes for data-efficient learning in robotics and control", PAMI-2015
- PILCO (Probabilistic Inference for Learning COntrol)
- A model based policy search method, considering the model uncertainty while learning models. PILCO Uses Gaussian processes as a probabilistic model.

PILCO Algorithm: High-level steps

1. Probabilistic model for transition function f (system identification)
2. Compute long-term predictions $p(x_1|\theta), ..., p(x_T|\theta)$
3. Policy improvement
4. Apply controller

1. Model Learning (System Identification)

Model learning problem: Find a function $f : x \mapsto f(x) = y$

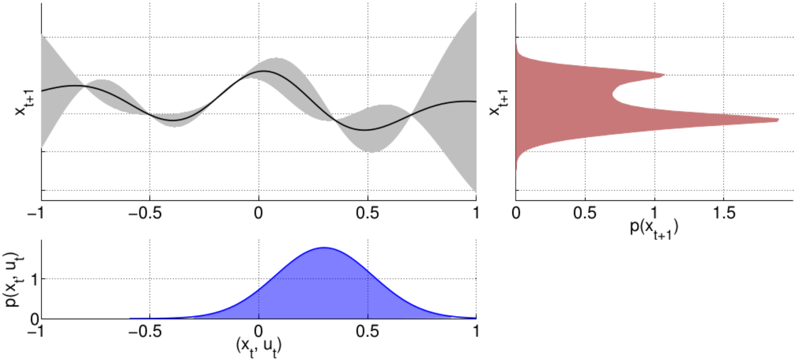


Distribution over plausible functions

Express uncertainty about the underlying function to be robust to model errors. Posterior GP prediction $p(x_{t+1}|x_t, u_t)$

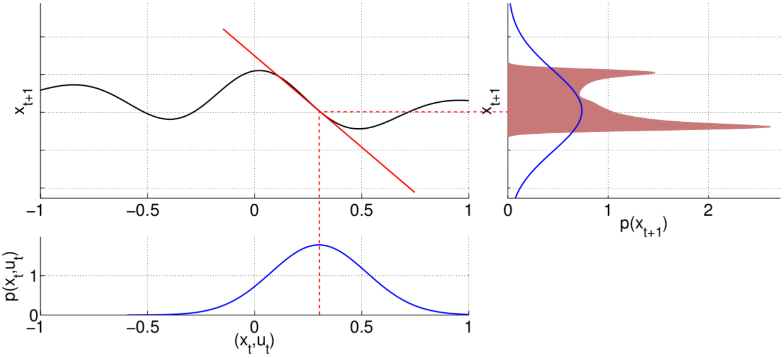
2. Long-Term Predictions (Policy evaluation)

$$\boxed{p(\boldsymbol{x}_{t+1}|\boldsymbol{\theta})} = \iiint \underbrace{\boxed{p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)}}_{\text{GP prediction}} \underbrace{\boxed{p(\boldsymbol{x}_t, \boldsymbol{u}_t|\boldsymbol{\theta})}}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})} \,\mathrm{d}f \,\mathrm{d}\boldsymbol{x}_t \,\mathrm{d}\boldsymbol{u}_t$$



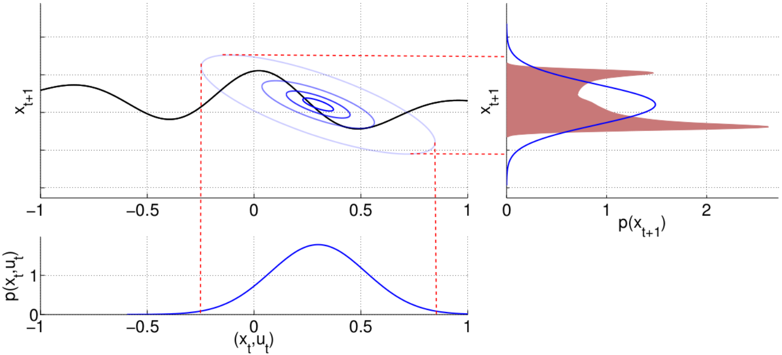
2. Long-Term Predictions (Policy evaluation)

Approximating $p(x_t + 1)$ (red in the last image) by a Gaussian.
Using : 1. linearization of the posterior mean function



2. Long-Term Predictions (Policy evaluation)

Approximating $p(x_t + 1)$ (red in the last image) by a Gaussian.
Using : 2. Moment matching



2. Long-Term Predictions (Policy evaluation)

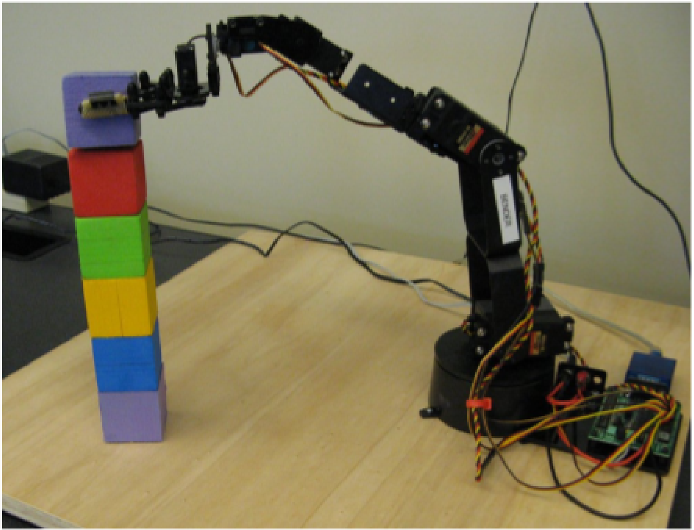
To evaluate expected long term cost $J(\theta)$ we choose cost function c such that inner the integral can be computed analytically:

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}[c(x_t)|\theta] = \sum_{t=1}^{T} \int c(x_t)\mathcal{N}(x_t|\mu_t, \Sigma_t)dx_t$$

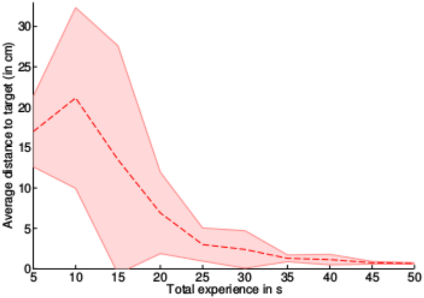
3. Policy Improvement and 4. Apply controller

- Analytically compute $\frac{dJ^\pi(\theta)}{d\theta}$ for a gradient based policy search.
- We use standard gradient based optimizer (e.g. BFGS) to compute θ^*
- Apply controller

Learning to Control a Low-Cost Manipulator



Learning to Control a Low-Cost Manipulator



(a) Average learning curve (block-stacking task). The horizontal axis shows the learning stage, the vertical axis the average distance to the target at the end of the episode.

PILCO algorithm

Algorithm 1 PILCO

1: *Define* policy's functional form: $\pi : z_t \times \psi \to u_t$.
2: *Initialise* policy parameters ψ randomly.
3: **repeat**
4: *Execute* system, record data.
5: *Learn* dynamics model.
6: *Predict* system trajectories from $p(X_0)$ to $p(X_T)$.
7: *Evaluate* policy:
$$J(\psi) = \sum_{t=0}^{T} \gamma^t \mathbb{E}_X[\text{cost}(X_t)|\psi].$$
8: *Optimise* policy:
$$\psi \leftarrow \arg\min_{\psi} J(\psi).$$
9: **until** policy parameters ψ converge

PILCO problems

- PILCO relies on Gaussian processes, which work extremely will with small amount of low dimentional data, but **scale cubically** with the number of trials, so PILCO is hard to scale to high dimensional observation spaces.
- PILCO doesn't consider temporal correclation between successive state transitions. This means that PILCO **underestimates state uncertainty at future time steps.** which can lead to diminished performance.

Deep PILCO

- Deep PILCO (Improving PILCO with Bayesian Neural Network Dynamics Models)
  (Yarin Gal and Rowan Thomas McAllister and Carl Edward Rasmussen - 2016)
- DeepPILCO :
  Gaussian processes $\rightarrow$ Bayesian deep dynamic model

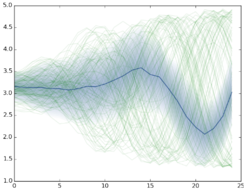
Deep PILCO idea

- Deep PILCO uses deep neural network capable of scaling to high dimensional observation spaces.
- The dynamic model that rely on DNN should maintain the probabilistic nature of the GP, capturing :
  1. Output uncertainty
  2. Input uncertainty

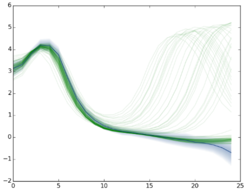
1. Output uncertainty solution

- Simple NN models cannot express output model uncertainty.
- Using Bayesian probabilistic equivalent to the NN - the **Bayesian neural network (BNN)**
- Represent the model uncertainty with an **approximate** posterior distribution over the weights of the BNN.
- **Dropout** can be interpreted as a variational Bayesian approximation. and can be used to approximate posterior distribution.
- Uncertainty in weights induces prediction uncertainty.

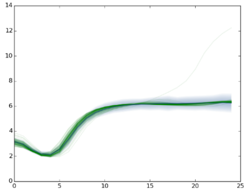
1. Output uncertainty solution - Dropout



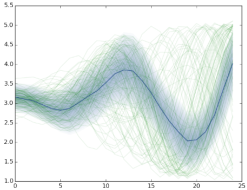
(a) Dropout $= 0$ (MAP)



(b) Dropout $= 0.05$



(c) Dropout $= 0.1$

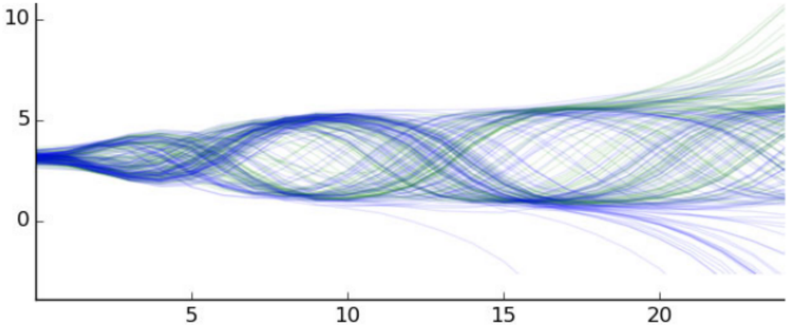


(d) Dropout $= 0.2$

2. Input uncertainty solution

- Handle input uncertainty = plan under dynamic uncertainty. Predict system trajectories from $p(x_0)$ to $p(x_T)$
- PILCO propagates state distributions through the dynamic model (GP) analytically, which cannot be done with NNs.
- The solution for this problem was using **particle methods**. Moment matching the output distribution at each time step, is used to forcing unimodal fit, where the algorithm penalizes policies that cause the predictive state to bifurcate.

2. Input uncertainty solution - Particle method



PILCO algorithm

Algorithm 1 PILCO

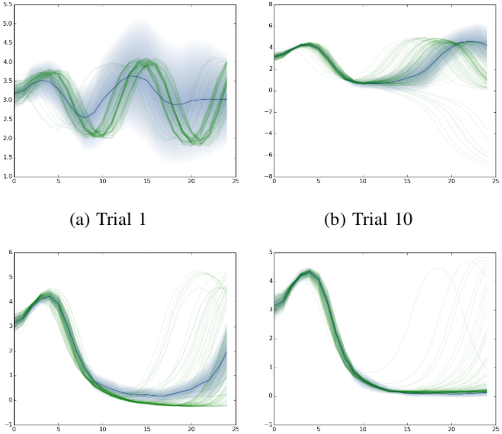
Define policy's functional form: $\pi : z_t \times \psi \to u_t$.
Initialise policy parameters ψ randomly.
repeat
  Execute system, record data.
  Learn dynamics model.
  Predict system trajectories from $p(X_0)$ to $p(X_T)$.
  Evaluate policy:
$$J(\psi) = \sum_{t=0}^{T} \gamma^t \mathbb{E}_X[\text{cost}(X_t)|\psi].$$
  Optimise policy:
$$\psi \leftarrow \arg\min_{\psi} J(\psi).$$
until policy parameters ψ converge

DeepPILCO modification

Algorithm 2 Step 6 of Algorithm 1: *Predict* system trajectories from $p(X_0)$ to $p(X_T)$

1: *Define* time horizon T.
2: *Initialise* set of K particles $x_0^k \sim P(X_0)$.
3: **for** $k = 1$ to K **do**
4: Sample BNN dynamics model weights W^k.
5: **end for**
6: **for** time $t = 1$ to T **do**
7: **for** each particle x_t^1 to x_t^K **do**
8: Evaluate BNN with weights W^k and input particle x_t^k, obtain output y_t^k.
9: **end for**
10: Calculate mean μ_t and standard deviation σ_t^2 of $\{y_t^1, ..., y_t^K\}$.
11: Sample set of K particles $x_{t+1}^k \sim \mathcal{N}(\mu_t, \sigma_t^2)$.
12: **end for**

DeepPILCO results on cartpole



(a) Trial 1

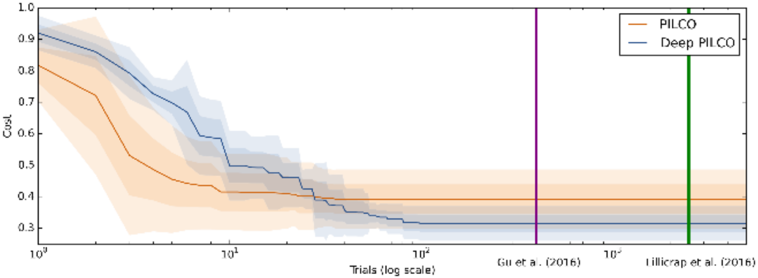
(b) Trial 10

(c) Trial 20

(d) Trial 26

Results continued

- DeepPILCO as a Bayesian approach to data efficient deep RL.
- Comparing DeepPILCO to DDPG and NAF (Normalized Advantage Functions)



Thanks!