TRAINING NEURAL NETWORK POLICIES WITH PROBABILISTIC MODEL-BASED RL

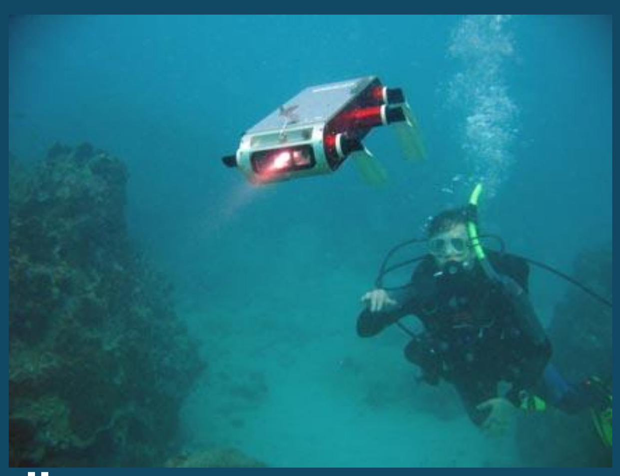
Juan Camilo **Gamboa Higuera**, David **Meger** and Gregory **Dudek**
{gamboa, dmeger, dudek}@cim.mcgill.ca

2nd Bayesian Deep Learning Workshop at NIPS 2017

McGill
School of Computer Science

Learning Controls in Robotics

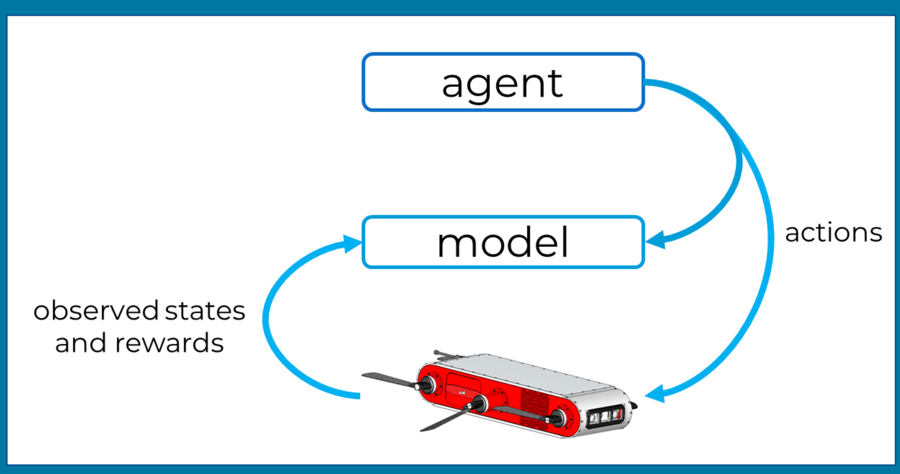
Automatically **synthesize controllers** for motor tasks on robots **deployed in the field**



Challenges:

- Collecting experience data is **expensive**
  - Minimize required experience (**data-efficiency**)
  - Re-use data across tasks
  - Minimize idle time between trials

Probabilistic Model-Based RL



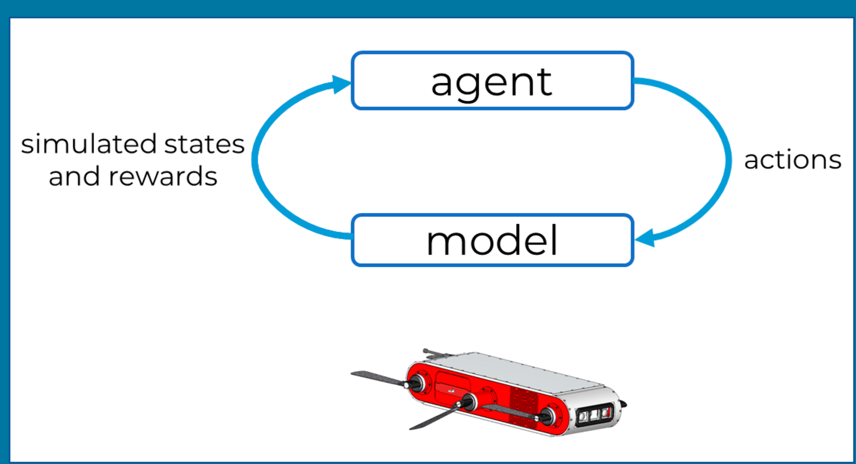
Execute a policy to gather **experience**.

Use experience for **model fitting**

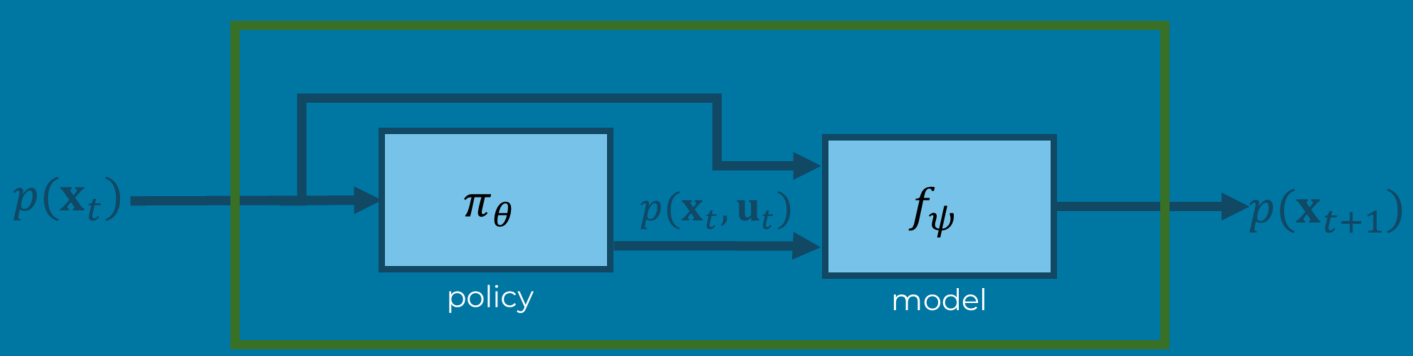
Probabilistic models learn a distribution over **plausible dynamics** (**model uncertainty**)

Use model to **simulate** experience and estimate **policy gradients**

Minimize **expected accumulated cost** over **dynamics model distribution**



Simulations via rollouts



Objective

$$J(\theta) = \mathbb{E}_\tau\left\{\sum_{t=0}^{H} c(\mathbf{x}_t)\right\} \approx \sum_{t=0}^{H} \mathbb{E}_{\mathbf{x}_t}\{c(\mathbf{x}_t)\}$$

PILCO [1]

- **Gaussian Process** Regression for dynamics
- Demonstrated with linear and **RBF** policies

Deep-PILCO [2]

- **MC-Dropout** for dynamics
- Demonstrated with **RBF** policies

Both methods model $p(\mathbf{x}_t)$ as Gaussian distributions via **moment-matching**

Problems

- Due to their **locality**, RBF policies **do not scale** to higher dimensional state spaces
- Deep-PILCO requires **hand-tuning** the dropout parameters
  - Increases the required experience
- Deep-PILCO shown to outperform PILCO on cart-pole task, but required more trials (**less data-efficient**)

1. M. P. Deisenroth, D. Fox, and C. E. Rasmussen. Gaussian processes for data-efficient learning in robotics and control (2015)
2. Y. Gal, R. McAllister, and C. E. Rasmussen. Improving PILCO with Bayesian neural network dynamics models (2016)
3. A. Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs (2000)
4. Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks (2013)
5. K. Neklyudov, D. Molchanov, A. Ashukha, and D.Vetrov. Structured Bayesian pruning via log-normal multiplicative noise (2017)
6. D. Meger, J. C. Gamboa-Higuera, A. Xu, G. Dudek. Learning legged swimming gaits from experience (2015)

Code available at https://github.com/juancamilog

Training NN controllers with Deep-PILCO

Reducing variance with CRNs

- Combine the Deep-PILCO algorithm with PEGASUS[3] policy evaluation

Input: cost c, dynamics model $p(f)$, initial state distribution $p(x_0)$, parametric policy π_θ

Sample k dynamic models $f^{(k)} \sim p(f)$
Sample $k(H-1)$ random vectors $\mathbf{z}_t^{(k)} \sim \mathcal{N}(0, I)$
for N optimization iterations
  Sample initial set of particles $\mathbf{x}_0^{(k)} \sim p(\mathbf{x}_0)$
  for $t = 1$ to H :
    Evaluate policy $\mathbf{u}_t^{(k)} = \pi_\theta\left(\mathbf{x}_t^{(k)}\right)$
    Propagate state $\mathbf{y}_{t+1}^{(k)} = f^{(k)}\left(\mathbf{x}_t^{(k)}, \mathbf{u}_t^{(k)}\right)$
    Compute mean $\boldsymbol{\mu}_{t+1}$ and covariance $\boldsymbol{\Sigma}_{t+1}$ of $\mathbf{y}_{t+1}^{(k)}$
    Resample $\mathbf{x}_{t+1}^{(k)} = \boldsymbol{\mu}_{t+1} + \boldsymbol{\Sigma}_{t+1}^{1/2}\mathbf{z}_{t+1}^{(k)}$
    Evaluate cost: $c_{t+1} = \mathbb{E}_{\mathbf{x}_{t+1}}\{c(\mathbf{x}_{t+1})\}$
  Update parameters $\theta \leftarrow \alpha\nabla_\theta(\sum_1^H c_t)$

Stochastic NN controllers

Training NNs **scales better** with state dimensions than RBFs

Training stochastic policies

- Sample one $\pi_\theta^{(k)} \sim p(\pi_\theta)$ for each dynamics model $f^{(k)} \sim p(f)$

Using stochastic policies

- Sample a new policy $\pi_\theta^{(k)} \sim p(\pi_\theta)$ at each time-step

Truncated log-normal multiplicative noise

- With MC-dropout, at the i-th layer of a NN with weights W and biases b

$$h_i = \boldsymbol{\sigma}(h_{i-1}W + b) \odot \epsilon_i$$
$$\epsilon_i \sim \mathrm{Bernoulli}(1-p)$$

- To avoid hand-tuning p, we experimented with

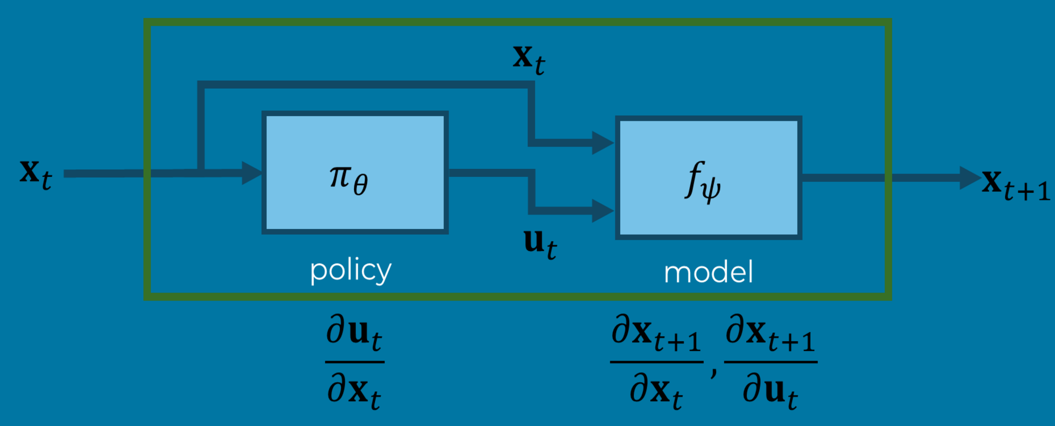
$$\epsilon_i \sim \mathrm{LogN}_{[a,b]}(\mu_i, \sigma_i^2)$$

regularized with a $\mathrm{LogU}_{[a,b]}$ prior [5].

- We set $a = -10$ and $b = 0$, which makes $0 < \epsilon_i < 1$
- We constrain $\sigma_i^2 < \mathrm{Var}\{\mathrm{U}_{[a,b]}\}$ to avoid "dead" units

Deep-PILCO rollouts as RNN

Rollouts with model equivalent to recurrent neural network



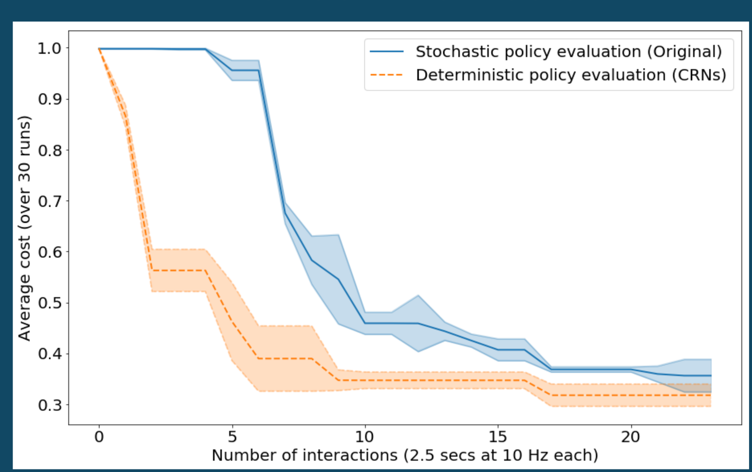
Susceptible to vanishing and exploding gradients [4] . Use clipping to stabilize learning

Results on Benchmark Tasks

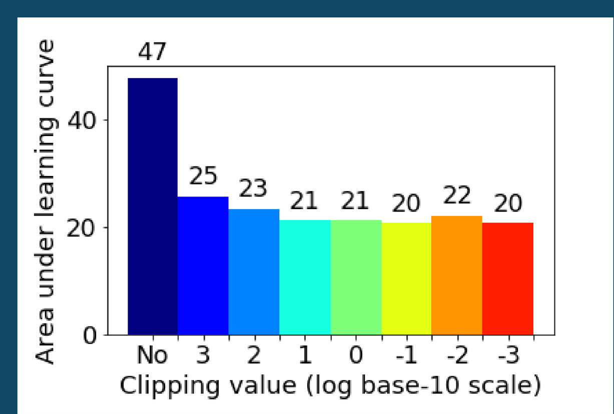
Using CRNs and clipping gradients **stabilize learning** and **improve data-efficiency**

These results were obtained on the cart-pole swing-up task with a single initial trial with controls sampled from an uniform distribution

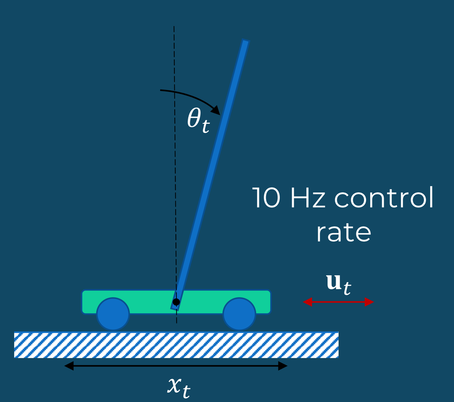
Effect of common random numbers

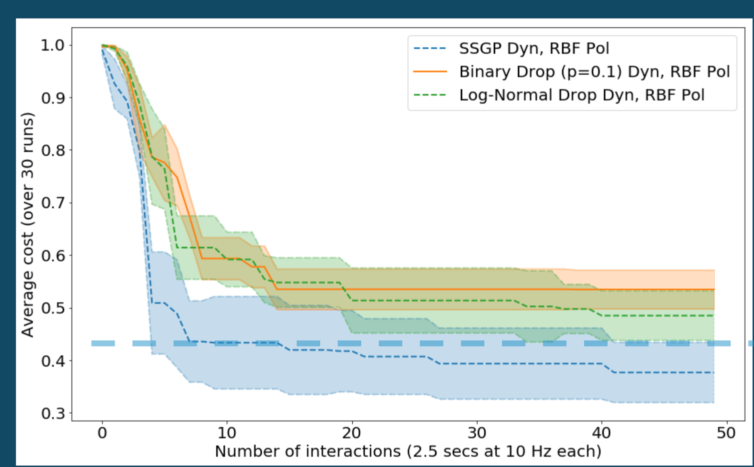


Effect of gradient clipping

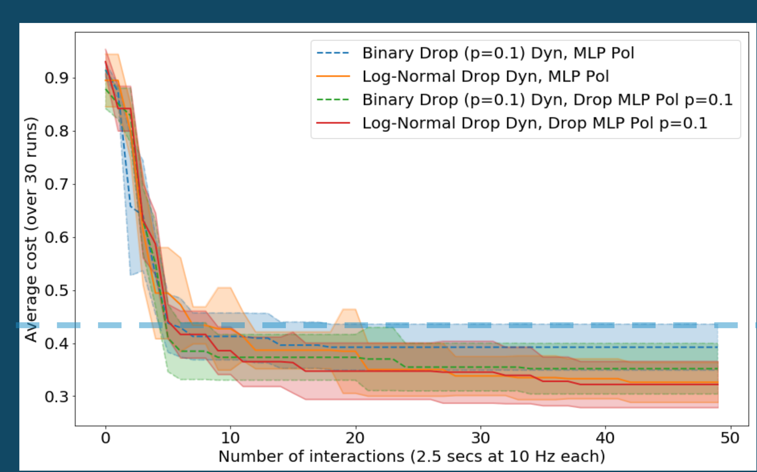


Cart-pole swing-up





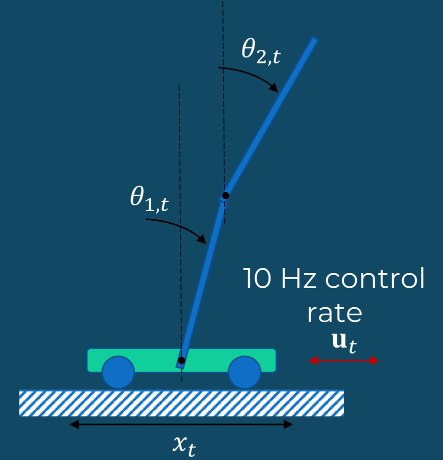
Using RBF Policies

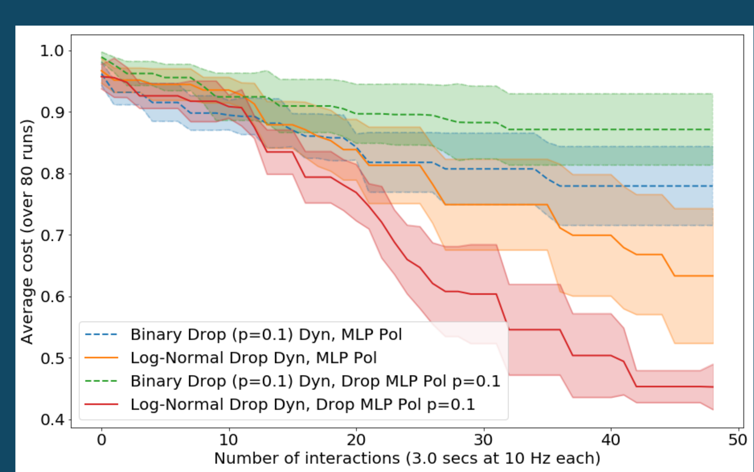


Using NN Policies

Cost for successful balancing with PILCO

Double pendulum on cart swing-up





Using NN Policies

Experience until task learned

PILCO*	Deep-PILCO*	Ours
17.5 s	~50 s	**17.5 s**

Cart-pole

PILCO*	Deep-PILCO	Ours
83 s (1120 samples at 13.3 Hz)	N/A	**126 s** (1260 samples at 10 Hz)

Double cart-pole

*Results reported by the authors in [1] and [2]

Summary and Outlook

- Demonstrated training of NN policies by Deep-PILCO with PEGASUS policy evaluation
- Data-efficiency comparable to PILCO
- Add **memory to policies**?
- Add **memory to dynamics**?
- **Alternatives for modelling uncertainty**?
- Ongoing experiments on underwater robot [6]