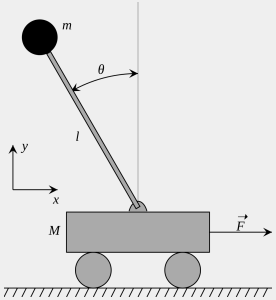
Improving PILCO with a Neural Network Gaussian Processes

Rishi Edupalli

Chantilly High School

April 2023



Definition

Field of understanding and building computer methods that learn, using data, to improve performance on some set of tasks.

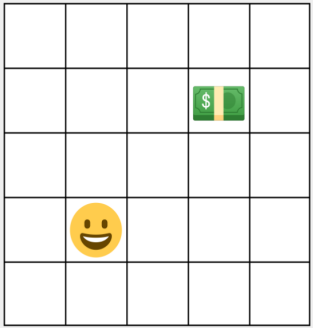
- AlphaZero
- Self Driving Cars
- ChatGPT

Reinforcement Learning

- Learner & Decision Maker: **Agent**
- Interacts with **environment**: continuously chooses **actions** A over time t under a **policy** π
- policy π defines dynamics, typically some probabilistic model over the set of actions.
- At each time step, Agent decides on action
- Receives numerical cost one time step later
- Agent attempts to minimize this cost

Trivial Example

- Goal: dollar
- Agent: grinning
- Actions: { N, E, S ,W }
- Policy: { 0.25,0.25,0.25,0.25 }
- Cost: Euclidean Distance from dollar and grinning



Model-Based Reinforcement Learning (MBRL)

- Model: Anything the agent can use to predict how the environment will respond to actions. Used to simulate the environment and produce simulated experience
- Planning: produces or improves policy by interacting with the modeled environment

PILCO: Probabilistic Inference for Learning Control

- MBRL Algorithm
- Can learn how to control robots in a physical system from scratch in a comparatively short period of time

$$\boldsymbol{x_{t+1}} = f(\boldsymbol{x_t}, \boldsymbol{u_t} + \epsilon)$$

- Vector describing state of the system at time t $\boldsymbol{x_t}$
- Control signal $\boldsymbol{u_t}$ (action)
- Unknown dynamics function f (model)
- Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$

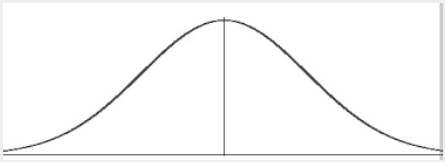
- Learn the vector of parameters θ of the agent/controller deterministic policy $\pi(x_t, \theta) = u_t$ that
- minimizes the sum of the average cost $c(x_t)$ over time T under policy π

$$J^{\pi}(\theta) = \sum_{t=0}^{T} \mathbb{E}[c(x_t)]$$

Gaussian (Normal) Distribution

$$\mathcal{N}(\boldsymbol{x_k}|\mu, \boldsymbol{\Sigma}) = \frac{\exp(-\frac{1}{2}(\mathbf{x}-\mu)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mu))}{\sqrt{(2\pi)^k \det(\boldsymbol{\Sigma})}}$$

- Mean Vector μ
- Covariance Matrix $\boldsymbol{\Sigma}$
- k-dimensional random vector $\boldsymbol{x_k}$

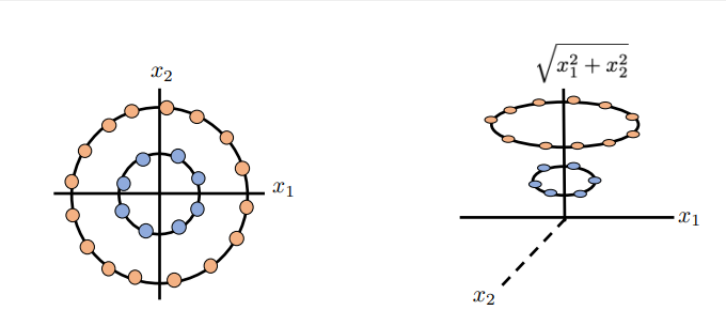


PILCO Algorithm

1. Randomly sample initial controller π_0 parameters $\theta \sim \mathcal{N}(0, \boldsymbol{I})$ and collect initial data by applying controller to robot
2. Learn the function f using data
3. Calculate $J(\theta)$ and use $\frac{dJ}{d\theta}$ to improve π_0
4. Use new policy to collect data
5. Repeat 2-4 until convergence to π^*

Kernel Regression

- PILCO uses Gaussian Process (GP) regression to learn the function f
- GP regression is a Kernel Regression method
- Nonlinear analogue of linear regression
- Main idea: transform nonlinear data into linear data by applying a transformation function
- Kernel measures distance in this Hilbert space



Step 2: Learning f via GP regression

- A GP is a probability distribution over possible functions f that can approximate our data
- They are defined by a mean and covariance function (kernel)
- PILCO's GP uses inputs (x_{t-1}, u_{t-1}) and outputs $\Delta t = x_t - x_{t-1} + \epsilon, \epsilon \sim \mathcal{N}(0, \Sigma_\epsilon)$ to predict f
- This is predicting the difference between states Δt given the control signal u_t and the previous state x_t
- We create a GP for every dimension of x
- PILCO assumes a mean function of zero and uses a Squared Exponential kernel

$$k(\tilde{x}, \tilde{x}') = \alpha^2 e^{-\frac{1}{2}(\tilde{x}-\tilde{x}')^T \Lambda^{-1} (\tilde{x}-\tilde{x}')}$$

$\alpha, \Lambda, \Sigma_\epsilon$ are hyper parameters (either set by user or learned)

Step 3: Calculate $J(\theta)$

$$\boldsymbol{x_{t+1}} = f(\boldsymbol{x_t}, \boldsymbol{u_t} + \epsilon)$$

- PILCO's GP provides one-step predictions
- We need to cascade these predictions over time T to calculate J^{π}
- The exact next state predictive distribution $p(\Delta t)$ can be represented as an integral however, it is not analytically computable

Step 3: Calculate $J(\theta)$ via Gaussian Moment Matching

- Assume that $p(\Delta t)$ can be approximated with a Gaussian $p(\Delta t) = \mathcal{N}(x_t|\mu_t, \Sigma_t)$
- Assuming the mean and covariance of these predictive distributions are known as μ_Δ and Σ_Δ (see [Deisenroth and Rasmussen, 2011]) we can write

$$\mu_t = \mu_{t-1} + \mu_\Delta$$
$$\Sigma_t = \Sigma_{t-1} + \Sigma_\Delta + cov[x_{t-1}, \Delta t] + cov[\Delta t, x_{t-1}]$$

Step 3: Improving π

- By cascading these predictions, an analytical form for the expected cost follows by the definition of $\mathbb{E}[x]$ at $t = 1, \ldots, T$ as

$$\mathbb{E}_{x_t}[c(x_t)] = \int c(x_t)\mathcal{N}(x_t|\mu_t, \Sigma_t)dx_t$$

- The $\int$ is a mathematical operation that returns the accumulated (net) change of a function
- We can improve the policy π with $\frac{dJ}{d\theta}$. This is a mathematical function that gives us the rate of change of J w.r.t. to the controller parameters θ.

PILCO and DeepPILCO Issues

PILCO

- Not really good at learning non-smooth dynamics
- Running time scales cubically w.r.t T and n

DeepPILCO

- Replaced GP with a Bayesian Neural Network (BNN)
- Easily scales to more complex systems
- Needs more data to learn

A 1-Hidden Layer Neural Network

- Data x, y
- Neural network f, Matrix A, Matrix B, element-wise activation function ϕ
- $f(x) = A\phi(Bx)$

Training A 1-Hidden Layer Neural Network

- Minimize loss $L(A, B) = \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - A\phi(Bx^{(i)}))^2$
- Initial Parameters $A^{(0)}$, $B^{(0)}$, Learning rate α
- Update A, B via Gradient Descent

$$A^{t+1} = A^{(t)} - \alpha \frac{\partial L}{\partial A}$$
$$B^{t+1} = B^{(t)} - \alpha \frac{\partial L}{\partial B}$$

- If we freeze B and sample it from a probability distribution, after some math, training a Neural Network is essentially kernel regression with $K(x, \bar{x}) = \langle \phi(Bx), \phi(B\bar{x}) \rangle$

$$L(A) = \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - A\phi(Bx^{(i)}))^2$$

$$A^{t+1} = A^{(t)} - \alpha \frac{\partial L}{\partial A}$$

- Initializing $B \sim \mathcal{N}(0, I)$ and letting the dimensions of $A, B \to \infty$, our neural network converges to a Gaussian Process and is called an NNGP.

Research Question & Hypothesis

Research Question

Does replacing the GP in the PILCO algorithm with an NNGP provide gains in performance? Specifically, in the cart-pole swing-up problem?

Hypothesis

NNGP-PILCO will have at least equivalent performance to PILCO

Intended Method

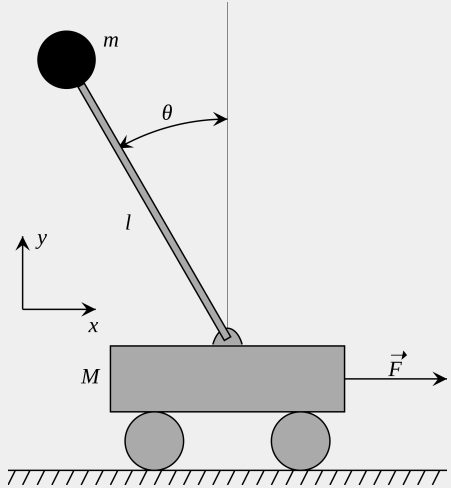
- Implement NNGP-PILCO and PILCO and record their performance at a reinforcement learning method
- Performance measure: Graph the cost over time

Cart-Pole Swing-Up

- $x = [x, \dot{x}, \theta, \dot{\theta}]$
- $x_{\text{target}} = [0, 0, \pi, 0]$
- $u \in [-10N, 10N]$
- $c(x_t) = 1 - e^{-\frac{||x_{\text{target}} - x_t||^2}{\sigma_c^2}}$
- $\theta = \{w_i, \Lambda, \mu_i\}$

$$\pi(x, \theta) = \sum_{i=0}^{n} w_i \phi_i(x)$$

$$\phi_i(x) = e^{-\frac{1}{2}(x-\mu_i)^T \Lambda^{-1} (x-\mu_i)}$$



Implementation Details

PILCO was implemented in Python using Jax and GPJax. Planned to implement NNGP-PILCO with Neural Tangents.

$$k(\tilde{x}, \tilde{x'}) = \alpha^2 e^{-\frac{1}{2}(\tilde{x}-\tilde{x'})^T \Lambda^{-1} (\tilde{x}-\tilde{x'})}$$

$$k(x,x') = e^{-\gamma(x-x')^T(x-x')}$$

$$\gamma = \frac{1}{2\Lambda^2}$$

Parameter	Value
T	25
n	50
Λ	**Learned via MLL**
α^2	1
ϵ	0

Table: Hyperparameter Settings

PILCO WAS FAILING TO LEARN GOOD ROLLOUTS WHEN $\alpha^2 = 1$

Actual x over time

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ -0.693974 \\ 0 \\ -2.1513193 \end{bmatrix} \rightarrow \begin{bmatrix} -0.0693974 \\ -1.4428222 \\ -0.21513192 \\ -2.321429 \end{bmatrix}$$

Predicted x over time, 10 MLL Optimizations

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} -4472381.692014483 \\ 77024158285233.61 \\ 10135887.13921894 \\ -208090469534488.7 \end{bmatrix} \rightarrow \begin{bmatrix} -32133746267120.06 \\ 1.8241505722521188e+24 \\ -1104014051687.7798 \\ -4.1218150782117 94e+23 \end{bmatrix}$$

Predicted x over time, 100 MLL Optimizations

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} -2268761.5671538827 \\ -22759325354304.797 \\ 5146873.879487108 \\ -58380499596828.28 \end{bmatrix} \rightarrow \begin{bmatrix} -19892448425295.797 \\ 5.0102642482095 29e+23 \\ -683567062744.3505 \\ 1.10810370552241 04e+24 \end{bmatrix}$$

Predicted x over time, 1000 MLL Optimizations

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0.0 \\ 0.27953227024835475 \\ 0.0 \\ 0.8665500377698998 \end{bmatrix} \rightarrow \begin{bmatrix} 0.0 \\ 0.27953227024835475 \\ 0.0 \\ 0.8665500377698998 \end{bmatrix}$$

Predicted x over time, 10000 MLL Optimizations

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} -6349.973886719554 \\ -597128281.0319556 \\ 20480.363992728126 \\ 622629532.6654978 \end{bmatrix} \rightarrow \begin{bmatrix} 5142948.250507345 \\ -743316740545.9059 \\ 959484.2719658513 \\ 1957064455579.111 \end{bmatrix}$$

Predicted x over time, 50000 MLL Optimizations

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 81.52807585224842 \\ -501197928.971957 \\ -21691.286512553186 \\ -1115504489.6414776 \end{bmatrix} \rightarrow \begin{bmatrix} -646442364.843772 \\ 86651659452032.06 \\ -20333429.403938837 \\ 266062367021231.88 \end{bmatrix}$$

Analysis

- Sensible: α^2 is a scaling factor in the context of the kernel.
- As increased accuracy of Λ will not compensate, α^2 is the culprit.

Conclusion

- The lengthscale parameter of a GP is not enough to encode information about both μ_Δ and Σ_Δ in the PILCO algorithm
- Conjecture: NNGP-PILCO will not provide gains in performance with PILCO when implemented with the dual activation of the RBF kernel as it is not possible to account for the variance.

Limitations

- NNGP-PILCO might work with a different Kernel
- May have been an other error in PILCO.

Thanks!

References I


Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018).
JAX: composable transformations of Python+NumPy programs.


Deisenroth, M. P. and Rasmussen, C. E. (2011).
PILCO: A model-based and data-efficient approach to policy search.
In Getoor, L. and Scheffer, T., editors, *ICML*, pages 465–472. Omnipress.


Gal, Y., McAllister, R., and Rasmussen, C. E. (2016).
Improving PILCO with Bayesian neural network dynamics models.
In *Data-Efficient Machine Learning workshop, International Conference on Machine Learning*.


Han, I., Zandieh, A., Lee, J., Novak, R., Xiao, L., and Karbasi, A. (2022).
Fast neural kernel embeddings for general activations.
In *Advances in Neural Information Processing Systems*.

References II

- Hron, J., Bahri, Y., Sohl-Dickstein, J., and Novak, R. (2020).
  Infinite attention: Nngp and ntk for deep attention networks.
  In *International Conference on Machine Learning*.
- Murphy, K. P. (2022).
  Probabilistic Machine Learning: An introduction.
  MIT Press.
- Murphy, K. P. (2023).
  Probabilistic Machine Learning: Advanced Topics.
  MIT Press.
- Novak, R., Sohl-Dickstein, J., and Schoenholz, S. S. (2022).
  Fast finite width neural tangent kernel.
  In *International Conference on Machine Learning*.

References III

Novak, R., Xiao, L., Hron, J., Lee, J., Alemi, A. A., Sohl-Dickstein, J., and Schoenholz, S. S. (2020).
Neural tangents: Fast and easy infinite neural networks in python.
In *International Conference on Learning Representations*.

Piedrahita, D. (2017).
Cart-pole control.

Pinder, T. and Dodd, D. (2022).
GPJax: A gaussian process framework in jax.
Journal of Open Source Software, 7(75):4455.

Radhakrishnan, A. (2022).
6.s088 modern machine learning: Simple methods that work lecture notes.

Sohl-Dickstein, J., Novak, R., Schoenholz, S. S., and Lee, J. (2020).
On the infinite width limit of neural networks with a standard parameterization.

References IV


Sutton, R. S. and Barto, A. G. (2018).
Reinforcement Learning: An Introduction.
The MIT Press, second edition.

Improving PILCO with a Neural Network Gaussian Processes

Rishi Edupalli

Chantilly High School

April 2023

