Model Based Reinforcement Learning for Learning Control of A Double Integrator

Rushikesh Sankhye
Department of Mechanical Engineering, College of Engineering, Northeastern University, Boston, USA

1 Abstract

Generate a time optimal speed profile for any non-linear dynamics resulting in a non-linear controller without any expert knowledge using a Model Based Reinforcement Learning algorithm called Probabilistic Inference for Learning Control (PILCO) This control policy is time optimal in generating inputs that produce rest-to-rest motion.

2 Method

Model Based Reinforcement Learning (RL). Unlike optimal control RL doesn't require expert knowledge.

Gaussian Process (GP) Regression was used to predict the transition dynamics for a double integrator system using GPyTorch simulated in a custom gym compatible environment. The double integrator model is embedded with uncertainties about the mass and damping. A RBF network is used to model the controller and minimize the expected trajectory cost Instead of using analytical gradients which can be tedious to calculate Adam optimizer from the PyTorch library is used. (Steiner, et al., 2019) An exponentially saturated cost function is used in this project. The cost used for the system includes a discounting factor as opposed to the general PILCO cost. Additionally, exploration is introduced to the cost function by including the standard deviation of the cost.

3 Theoretical Background –

3.1 Gaussian Process Regression

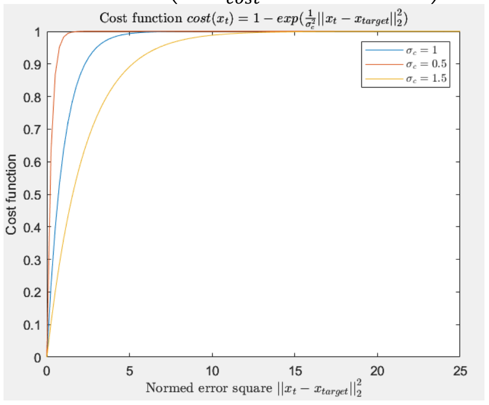
There is a model bias where the few samples we collect in the real world do not fully summarize the transition function and our uncertainty about it. Gaussian Processes (GP) (Williams & Rasmussen, 2006) help summarize this uncertainty by using a Bayesian framework to estimate the posterior distribution of the output state given an ordered input where in a dynamical model is

$$x_t = f(x_t - 1, u_t - 1)$$
$$y_t = x_t + \text{noise}$$

The function we are trying to estimate is f, the transition model which may be a non-linear function and the input data is D = $\{x_i, u_i\}$. The output y is the delta change in position.

3.2 Cost

$$Cost(x_t) = 1 - exp\left(-\frac{1}{2\sigma_{cost}^2}\left|\left|xt\text{–}\,xtarget\right|\right|_2^2\right) \in [0,1]$$



Here, σ_c is the cost width.

3.3 Policy Evaluation:

$$c(x_t) = 1 - exp\left(-\frac{1}{2}(x-z)^T W(x-z)\right)$$
$$\in \quad [0,1]$$
$$E_{x_t}[c(x_t)] = \int c(x_t)\mathcal{N}(x_t|\mu_t, \Sigma_t)dx_t$$

where, x ~ $\mathcal{N}(\mu, \Sigma)$, z is the target state and W plays the role of a precision matrix. We set the influence of the velocity state variable in the cost to zero by setting the corresponding value σ_{vel} in W to zero in order to only penalize position, since the two are correlated.

$$W = \begin{bmatrix} \sigma_{pos} 0 \\ 0 \ \sigma_{vel} \end{bmatrix}$$
$$J^{\pi}(\theta) = \sum_{t=0}^{T} E_{x_t}[c(x_t)]$$

3.4 System -

A double integrator was chosen as the system to learn a controller

$$x_{t+1} = Ax + Bu$$
$$A = dt\begin{bmatrix} 1 & 1 \\ 0 & 1 - \frac{c}{m} \end{bmatrix}$$
$$B = dt\begin{bmatrix} \frac{1}{2}dt \\ \frac{1}{m} \end{bmatrix}$$
$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}$$

where, q is the position of the system and $\dot{q}$ is the velocity of the system

4 Optimal Control Formulation -

Consider the double integrator in our problem

$$\ddot{q} = u, \qquad |u| \le 1$$

Our goal is to design a policy $u = \pi(x,t), where\ x = [q, \dot{q}]^T$ to get to the goal state defined as

$$x(T) = [q(T), \dot{q}(T)]^T$$
$$where, T\ is\ the\ terminal\ time$$

Since our goal is to displace our system to the target position given by $q(T)$ from $q(0)$ in a rest to rest the velocities at the boundaries will be $\dot{q}(0) = 0$ and $\dot{q}(T) = 0$

We want to optimize this problem with a minimum time formulation.

$$\boxed{\begin{gathered} min_{\pi}J = \overbrace{h(x(T))}^{\text{final cost} = 0} + \int_0^T \overbrace{g(x,u)}^{\text{instantaneous cost} = 1} dt \\ = 0 + \int_0^T 1dt = T \\ \text{st} \quad x(0) = [q(0), \dot{q}(0)]^T, \quad \text{Inital conditon} \\ x(T) = [0,0]^T \quad \text{Final Condition} \\ \dot{x} = f(x,u) = Ax + Bu \quad \text{Dyanmics} \end{gathered}}$$

This problem has been studied in optimal control and is known as the Bang-Bang Control of a Double Integrator.

We need to first find the minimum time T^* that our system can achieve. The analytical solution for this system in continuous time is

$$I_{q,\dot{q}} = (\dot{q} < 0)\&(\frac{1}{2}\dot{q}^2 - q \le 0)|(\dot{q} < 0)\&(\frac{1}{2}\dot{q}^2 - q \le 0)$$

$$T^* = \begin{cases} \dot{q} + 2\sqrt{\frac{1}{2}\dot{q}^2 + q} & \text{if } I_{q,\dot{q}} = 1 \\ 2\sqrt{\frac{1}{2}\dot{q}^2 - q} - \dot{q} & \text{if } I_{q,\dot{q}} = 0 \end{cases}$$

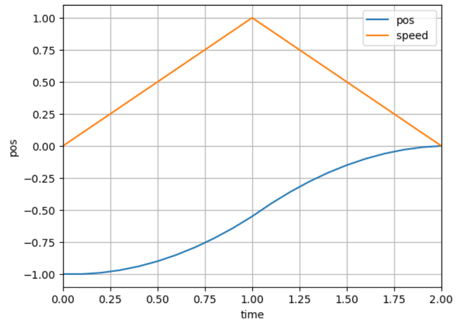
Taking m = 1 , c = 0 and initial conditions q = -1 and $\dot{q}$ = 0, we get the solution,

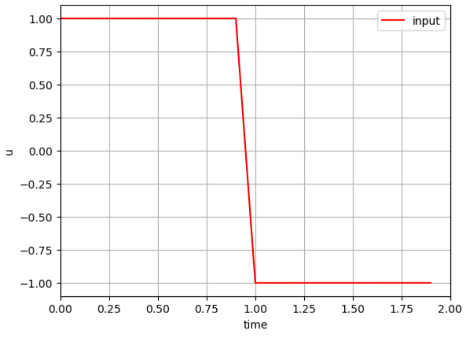
$$T = 2\sqrt{\frac{1}{2}\dot{q}^2 - q} - \dot{q} = 2\sqrt{1} = 2s$$

Solving the controlled optimization problem in the constrained optimization problem with

$$N = \frac{T}{dt} = \frac{2}{0.1} = 20$$

produces the following solution





5 RBF Controller -

This policy will be used to train our Radial Basis Function (RBF) controller on a test input

$$\tilde{\pi}(x_*) = \sum_{i=1}^{N} a_i \rho \left\| x_* - c_i \right\|_2^2$$

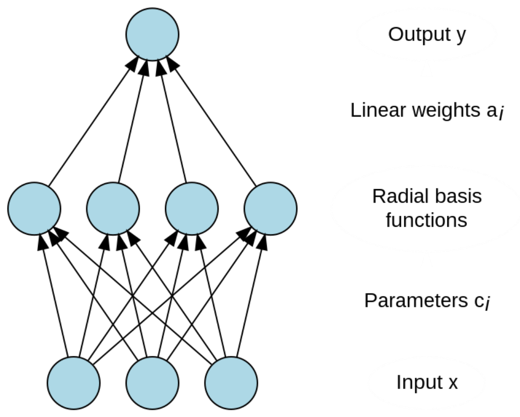
where, ρ is the radial basis function which is taken to be a Gaussian

$$\rho\left(\left\| x_* - c_i \right\|_2^2\right) = exp\left(-\beta_i \left\| x_* - c_i \right\|_2^2\right)$$

x_* and N is the number of support points or basis functions

The parameters $\theta = \{a_i, c_i, \beta_i\}, i = 1, .., N$ are optimized to minimize the policy loss $J^{\pi}(\theta)$

For N = 100, $\theta \in R^{300}$



The final layer at the end of this network is a sine squashing applied which limits the input within symmetric bound $\in [-u_{\max}, u_{\max}]$ and the policy is then given by

$$\pi(x_*) = u_{\max} sin\left(\tilde{\pi}(x_*)\right)$$

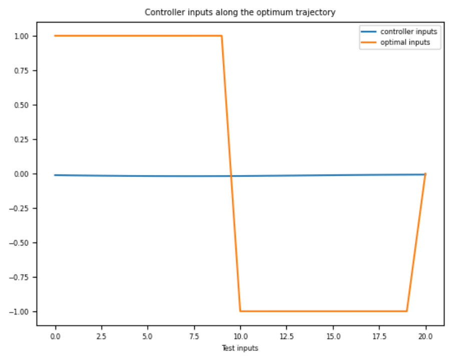
6 Loss function implementation

$$J^{\pi}(\theta) = \sum_{t=0}^{T} \gamma^{-t} E_{x_t}[c(x_t)]$$

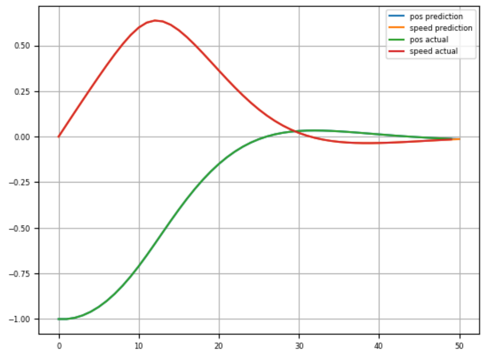
The loss function $J^{\pi}(\theta)$ includes a γ term for discounting which produces a 'discounting' effect except that it is penalizing the cost more towards the end of the trajectory.

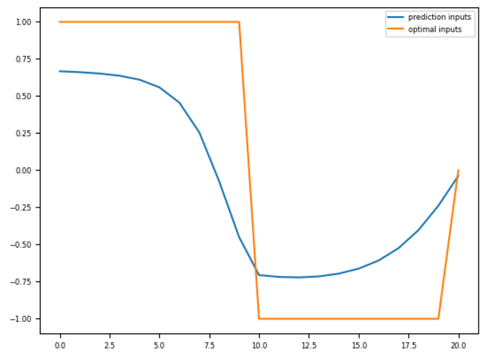
7 Results -

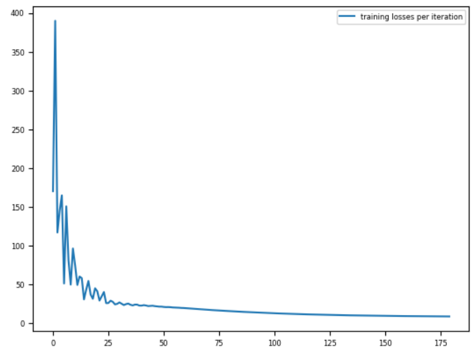
Code for generating the results is given in[1] The policy is randomly initalized as shown in below



After 180 policy optimization iterations were run with the model being updated every 12 iterations these were the results.







8 Conclusions

PILCO was implemented in PyTorch and GPyTorch. The algorithm was applied on a double integrator system to find the minimum time solution. The resulting controller was compared with results derived from optimal control.

While hard coding gradients requires careful attention and knowledge of the gradients, it considerably saves computation time. Training is done on a 12th Gen Intel(R) Core(TM) i7-12700H (2.30 GHz) with 16GB RAM. The algorithm using back propagated gradients that are automatically computed by PyTorch in contrast is easier to implement but due to the computational load takes much longer to converge. The decay of the losses slows down considerably after the first 50 iterations. If training was continued the solution would have converged to the true optimal solution. This was confirmed by pretraining the RBF network on a perturbed version of the optimal solution and confirming that it converges. Further studies would include addition of safety filters at the end of the trajectory which would avoid overshooting and therefore result in faster convergence.

[1] https://github.com/rsankhye/CS5180-PILCO

9 Bibliography

Deisenroth, M. P., & Rasmussen, C. E. (2011). *PILCO: A Model-Based and Data-Efficient Approach to Policy Search*. Retrieved 12 11, 2023, from http://mlg.eng.cam.ac.uk/pub/pdf/deiras11.pdf

Steiner, B., DeVito, Z., Chintala, S., Gross, S., Paszke, A., Massa, F., . . . Bai, J. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Retrieved 12 11, 2023, from https://nips.cc/conferences/2019/acceptedpapersinitial

Tedrake, R. (2008). *Underactuated Robotics: Learning, Planning, and Control for Efficient and Agile Machines.* Retrieved 12 11, 2023, from http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-832-underactuated-robotics-spring-2009/readings/MIT6_832s09_read_ch01.pdf

Thomaz, C. E., Feitosa, R. Q., & Veiga, A. (1998). *Design of radial basis function network as classifier in face recognition using eigenfaces*. Retrieved 12 11, 2023, from http://general-vision.com/pub3rdparty/3p_rbf_as_facerecoclassifier.pdf

Williams, C. K., & Rasmussen, C. E. (2006). *Gaussian Processes for Machine Learning.* The MIT Press. Retrieved 12 11, 2023, from http://www.gaussianprocess.org/gpml/

Adam — PyTorch 2.1 documentation. (n.d.). Retrieved December 11, 2023, from https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

Contributors to Wikimedia projects. (2023, July 25). Radial basis function network. Wikipedia. https://en.wikipedia.org/wiki/Radial_basis_function_network

Deisenroth, M. P. (2010). Efficient reinforcement learning using gaussian processes. KIT Scientific Publishing.

Liao, H.-C., Chou, H.-J., & Liu, J.-S. (2022). Velocity planning via model-based reinforcement learning: Demonstrating results on PILCO for one-dimensional linear motion with bounded acceleration. In Artificial Intelligence Annual Volume 2022. IntechOpen. http://dx.doi.org/10.5772/intechopen.103690

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

Tedrake, R. (2023). Underactuated Robotics (pp. 93–114). https://underactuated.csail.mit.edu