Toward Faster Reinforcement Learning for Robotics: Using Gaussian Processes

Ali Younes[1] and Aleksandr I. Panov[2,3](✉)

[1] Bauman Moscow State Technical University, Moscow, Russia
ay20-5-1994@hotmail.com
[2] Artificial Intelligence Research Institute,
Federal Research Center "Computer Science and Control"
of the Russian Academy of Sciences, Moscow, Russia
pan@isa.ru
[3] Moscow Institute of Physics and Technology, Moscow, Russia

Abstract. Standard robotic control works perfectly in case of ordinary conditions, but in the case of a change in the conditions (e.g. damaging of one of the motors), the robot won't achieve its task anymore. We need an algorithm that provide the robot with the ability of adaption to unforeseen situations. Reinforcement learning provide a framework corresponds with that requirements, but it needs big data sets to learn robotic tasks, which is impractical. We discuss using Gaussian processes to improve the efficiency of the Reinforcement learning, where a Gaussian Process will learn a state transition model using data from the robot (interaction) phase, and after that use the learned GP model to simulate trajectories and optimize the robot's controller in a (simulation) phase. PILCO algorithm considered as the most data efficient RL algorithm. It gives promising results in Cart-pole task, where a working controller was learned after seconds of (interaction) on the real robot, but the whole training time, considering the training in the (simulation) was longer. In this work, we will try to leverage the abilities of the computational graphs to produce a ROS friendly python implementation of PILCO, and discuss a case study of a real world robotic task.

Keywords: Robot learning · Reinforcement learning · Gaussian process · Data efficient

1 Introduction

The standard control methods in robotics are based on the dynamical model of the robot, and also on the model of the dynamics of the environment to build the needed closed loop control scheme [1–6]; in the real world to realize such methods for manipulators, we have to follow the following steps: (1) taking an observation of the environment using cameras or sensors (2) estimating the state

This work was supported by the Russian Science Foundation, project no. 18-71-00143.

© Springer Nature Switzerland AG 2019

G. S. Osipov et al. (Eds.): Artificial Intelligence, LNAI 11866, pp. 160–174, 2019.
https://doi.org/10.1007/978-3-030-33274-7_11

pan@isa.ru

of the robot and the task (e.g. position of the end-effector and the goal position) (3) planning the trajectory of motion of the end-effector to achieve the task (4) using low-level controllers (or force controller for harder tasks) to ensure following the planned path by minimizing the errors (5) sending the resulting commands to the joints of the robot. The errors which are occurred in each step, accumulated to produce a cumulative error making the control process hard to realize with desired accuracy.

The essence of the robot learning is to find a way to develop robotic behavior to a human's level behavior. Hence the Reinforcement Learning (RL) [6,7] seems to be the most viable way for robot learning, where the learning process depends on an agent taking actions, noticing the changes in the environment's state and the resulted reward of that action. The goal of RL is to learn the best possible policy to achieve a task by a trial and error hypothesis.

The state of the art deep reinforcement learning algorithms, which has tried to handle robotic tasks can be classified to two major classes: (1) model-free algorithms: (TRPO [9], PPO [10], DDPG [11]) which can learn to achieve the task after sampling training sets from interacting with environment, so we can consider the robot's model as a black-box (2) model-based algorithms ([12–14]): depends on a learned transition model of the environment. The model-free algorithms need days of training to learn basic robotic tasks. On the other hand, ordinary model-based algorithms can learn much faster (less than an hour), but mostly can't adapt to unforeseen situation (the learned model is no longer valid) such in case a damaged motor [15–17].

Model-based algorithms learns a state transition model, that represent how would the next state will be in case of taking an action, without knowing the dynamic model of the robot. When using deterministic models, the results of the RL depend on the accuracy of the model, and mostly it failed with unforeseen states.

In this paper we are interested in the idea of using probabilistic models in RL algorithms [18–21], to handle the uncertainty of model and reduce its training time. The approach uses a Gaussian process (1) its input will be a state x_t (the robot joints' angles and positions) and the control u, (2) the output will be the resulted state x_{t+1} (or the difference $x_{t+1} - x_t$). The reason to use the Gaussian process is its ability to learn from small data sets. After training the model we will use it to simulate the task (generate trajectories), and optimize the controller over that trajectories. We have chosen PILCO algorithm [22], which considered the most data efficient RL algorithm, we are interested in using of computational graphs to implement PILCO, and see how our work could scale to robotic tasks.

The following sections are structured as follows. First we will outline some preliminaries (Sect. 2), which will be a brief introduction to RL, GP, PILCO and computational graphs. Then we describe our work (Sect. 3) and experiments on a robotic task (Sect. 4). Then we discuss out the results (Sect. 5). Finally, we add a discussion (Sect. 6) and future work (Sect. 6).

pan@isa.ru

2 Preliminaries

2.1 Reinforcement Learning

Reinforcement learning is a part of the machine learning, which study how should the agent have to interact in its working space, in order to minimize (maximize) a long-term cost (reward) (see Fig. 1).

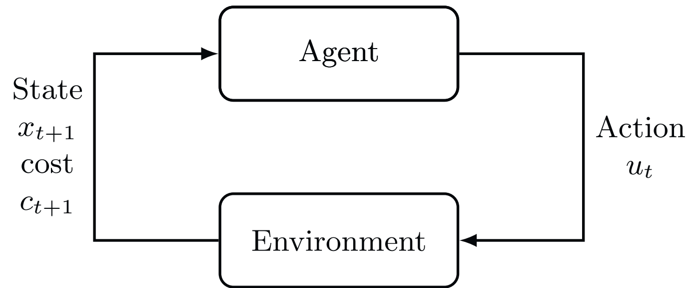

Fig. 1. Reinforcement learning paradigm

We represent the RL problem as a Markov Decision Process (MDP) at Fig. 2.

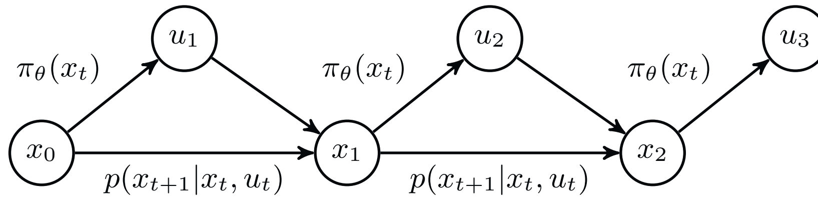

Fig. 2. Markov decision process for RL problem

Where x_t is the state, u_t - control (action), c - cost (reward), the transition function: $x_{t+1} = f(x, u_t) + \omega$, that we aim to learn in model based RL. The policy function, which gives the best action for each state after the training process (could be called as the controller): $u_t = \pi(x_t, \theta)$. The goal is to minimize the expected long-term cost:

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}[c(x_t)|\theta].$$

2.2 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

pan@isa.ru

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 mean function $m(.)$ and a covariance function (kernel) $k(.,.)$.

We will use an independent Gaussian process for each dimension (variable) of the output. It will describe how would be the next state beginning from the current state and implementing control signal u, $f : x \to f(x_t, u) = x_(t+1)$. The Gaussian process will learn using the data collected from the real robot, the data set consists of transitions x_t, u_t, x_{t+1}, c_t. And the learning process is a regression problem, so if we start from a prior

$$P(f|x) \sim \mathcal{N}(\mu, \Sigma).$$

We get a posterior

$$P(y_*|D, x) \sim \mathcal{N}(\mu_(y_*|D), \Sigma_(y_*|D))$$

after a training epoch (the process is called Bayesian inference) (see Fig. 3).

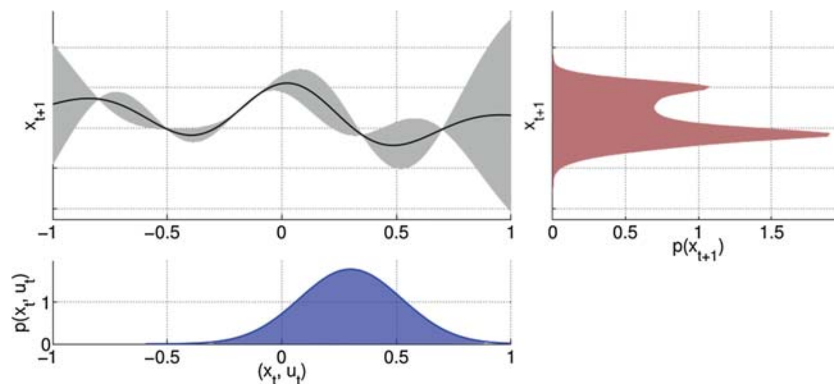

Fig. 3. GP prediction at uncertain input. The input distribution $p(x_t, u_t)$ (the blue curve and term in the equation) propagates though the GP model (the gray model), we obtain the expected distribution of the next state $p(x_{t+1})$ (Color figure online)

And we will use the learned model to make a long term prediction (build trajectories in the simulation steps), so if we make an action u in the state x, we map it through the Gaussian process to get the output as the probability of the next state. Using the formula:

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

The output distribution is irregular, so we use the moment matching algorithm to approximate it.

pan@isa.ru

2.3 PILCO Algorithm

PILCO (Probabilistic Inference for Learning COntrol) algorithm [22] is a model-based policy search reinforcement learning algorithm, which achieved unprecedented data-efficiency of several control benchmarks. PILCO is a model based algorithm, which means it consists of two alternating steps:

1. Interaction step in which we run the real robot (using a random policy in the first episode, and the optimized policy afterward), collect the roll-out's data, and train the Gaussian process model on the collected data.
2. Simulation step in which we have to:
   (a) Use the Gaussian process to build long-term predictions over a trajectory from $p(X_0)$ to $p(X_T)$.
   (b) After that compute the long term cost function:

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}[c(x_t)|\theta]$$

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

   (c) At the end use the computed cost to optimize the controller's parameters to minimize the cost, by using a line search algorithm based on the gradient of the cost function (L-BFGS-B algorithm):

$$\theta \leftarrow \arg\min_{\theta} J(\theta)$$

PILCO is summarized by Algorithm 1.

Algorithm 1. PILCO

1 Define a model and a policy
2 Collect a random roll-out, record data
3 **repeat**
4 learn the model
5 Collect trajectories using the model from $p(x_0)$ to $p(x_T)$
6 evaluate the policy $J(\theta) = \sum_{t=1}^{T} \mathbb{E}[c(x_t)|\theta]$
7 optimize policy $\theta \leftarrow \arg\min_{\theta} J(\theta)$
8 run the policy and collect data
until *task solved*;

Model bias is a problem that faced model-based algorithms, when selecting only a single dynamic model and assuming that model is the correct model, and hence the prediction errors in the model compound to produce a inaccurate long

pan@isa.ru

term predictions. PILCO uses Gaussian processes as a probabilistic models to avoid model bias, by considering all plausible dynamics models in prediction of the next states, i.e. give the model sufficient uncertainty. Which leads to a better results in terms of data efficiency.

2.4 Computational Graphs for Gaussian Process Regression

Computational graphs are directed graphs, in which the nodes are either variables or operations, and the edges define the inputs to each node (see Fig. 4).

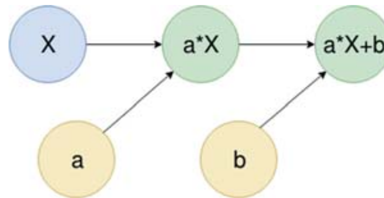

Fig. 4. Simple computation graph; the blue node is the input X, the yellow nodes are constants, and the green nodes are the operations (Color figure online)

There are two key strength of using computational graphs:

- It can be used to form a complex operations from simple operation.
- They enable automatic differentiation, which is needed in optimization.

We propose using computational graphs for Gaussian process regression (Algorithm 1, line 4), where the process of learning the model means fitting a probability model to the collected data, in other words; we start from a (1) prior distribution with zero mean function and an initial covariance function, (2) observe the collected data and compute the posterior distribution, and after that (3) learning the hyper-parameters (length-scales, signal variances and noise variances which define the covariance function) of the GP via evidence maximization.

The bottle-neck in this process, is in (1) the computation of the posterior over the data points, and (2) the differentiation which is needed in the process of evidence maximization.

The formulas to find the mean and the variance of the posterior:

$$m_t(x) = K(x, X_t)[K(X_t, X_t) + \sigma_\epsilon^2 I)]^{-1} y_t$$

$$k_t(x, x) = k(x, x) - K(x, X_t)[K(X_t, X_t) + \sigma_\epsilon^2 I)]^{-1} K(X_t, x)$$

where X_t is the observed inputs, x all possible input points, y_t observed outputs and σ_ϵ noise variance.

We can compute then evidence (log marginal likelihood):

$$\log p(y|X, \phi) = -\frac{1}{2} y^T K_y^{-1} y - \frac{1}{2} \log |K_y| - \frac{n}{2} \log 2\pi$$

where $K_y = K(X, X) + \sigma_\epsilon^2 I$ is the covariance matrix of the noisy outputs y.

pan@isa.ru

The hyper parameters of the GP:

$$\phi = (l, \sigma_f^2, \sigma_\epsilon^2)$$

length-scales l, signal variance σ_f^w, noise variance σ_ϵ^2.

The evidence maximization goal:

$$\hat{\phi} = \arg\max_{\phi} log(p(y|X, \phi))$$

The evidence maximization process, make uses of the partial derivatives of the log marginal likelihood with respect to the hyperparameters to find the combination that maximize evidence.

By representing the previous relations as a computational graph, we can (1) leverage the GPU by run the matrix operations on it, (2) use the automatic differentiation property of the computational graph instead of compute the derivatives analytically (see Fig. 5).

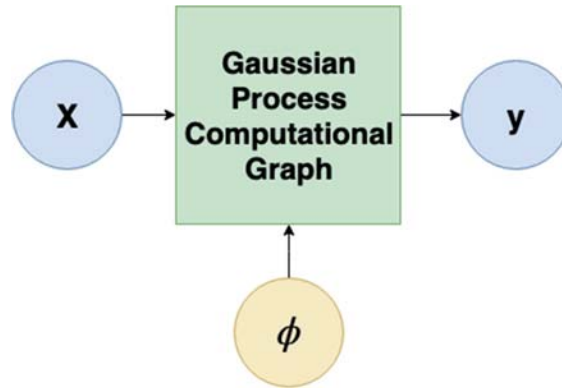

Fig. 5. High level representation of the computational graph used for Gaussian process regression

3 Problem Formulation

We consider using Gaussian process to achieve better data efficiency for reinforcement learning in robotic tasks. We will use PILCO as a base algorithm for our work (as it is the most data efficient RL algorithm). In this work, we start from two observations about PILCO:

– The Robot Operating System (ROS) [25] is an open source, flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Most of the robotics laboratories are using ROS in research experiments and projects, and it has been used widely in the industry, making it as the most powerful tool in the robotics community. ROS support C++ and Python only, but the official code release of PILCO [26] was written in Matlab, which makes it not compatible with ROS, So we decided to reproduce PILCO in Python to make it compatible with ROS.

pan@isa.ru

– The recent revolution of the deep learning relies on exploiting the computing power of the GPU, the father of the RL Richard Sutton has mentioned it in his "bitter lesson" [27]. PILCO reduces the amount of interaction time on the real robot, but it takes a relatively long time after that for inference and controller optimization. Reducing this time may give the algorithm the ability to learn in real time (or with a little latency), for robotics, it means the ability to adapt to unforeseen cases, which is a step toward the intelligent robot. We used GPflow library [23], which is a Gaussian process library that uses TensorFlow for its core computations and Python for its front. GPflow follows the computation graphs of the TensorFlow, which make the best use of the GPU power. That facts reduce the training time (especially when working with large scales).

We evaluated our implementation on a 7-DoF robotic arm task, in the OpenAI gym [24] to robotic environment. In the following, we will discuss the Experimental setup of this experiment, with an explanation that is needed to understand the points of interest, and sample of results with discussion.

4 Experimental Setup

We applied our implementation on a 7-DoF robotic arm (Fig. 6). We assume that we don't know any thing about the model of the robotic arm or the environment, we can just observe the coordinates of the end effector and joints' angles, and receive a reward (cost) after each movement. We can control the robotic arm by sending 7 control signals to each of its joints. For the algorithm, it is not needed to know what the state or the control signal represents, but to make our experiment more applicable to real world robots, we will constrain the control signal, and we will also constrain the length of the interaction phase.

Firstly we will define the following:

– **State:** the coordinates of the end effector and the joints' angles

$$X_e = [x_e, y_e, z_e, j_0, ..., j_7] \in \mathbb{R}^{10}$$

– **Target:** the coordinates of the target

$$X* = [x*, y*, z*] \in \mathbb{R}^3$$

– **Actions:** the control signal for each joint

$$U_j = [u_0, u_1, ..., u_7] \in \mathbb{R}^7$$

– **The cost function:**

$$c(x) = 1 - exp(-\frac{1}{2\sigma_c^2} d(X_e, X*)^2) \in [0, 1]$$

pan@isa.ru

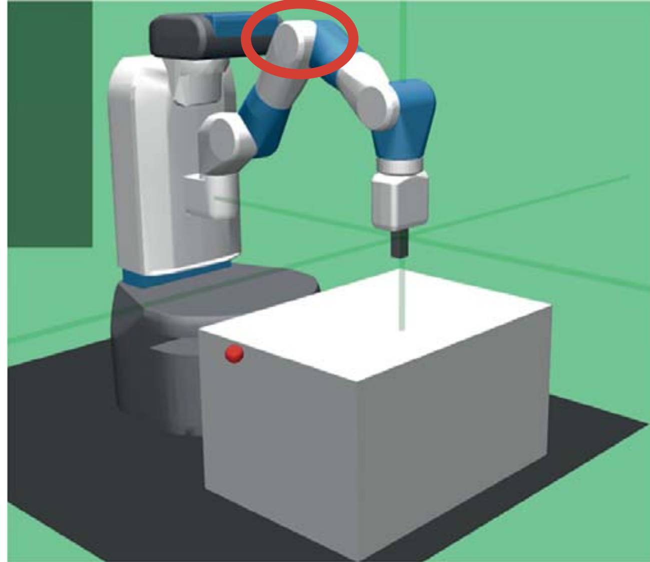

Fig. 6. A 7-Dof robotic arm's task in simulation, the control signal u is just a relative rotation angle for each joint, the state is the position of the end effector and joints' angles, the goal is to reach the red point. The joint which is surrounded by an ellipse, is the broken (unresponsive) joint in the second part in results (Color figure online)

- **The transition model** the model consists of an independent stochastic GP regression model for each variable of the output. In our test case, we have 10 GPs, each one takes the state and the control command as an input, and the output is resulted difference of one of the state variables (Fig. 7a).
- **Controller** we have used RBF (Radial Basis Function) as a controller. We can use the RBF controller as a deterministic GP regression model, by considering it like that, we exploit the multi output GP class that we have already used for the transition model (Fig. 7b).

5 Experimental Results

5.1 Classic Reaching Task

As we are interested in an implementation that could be applicable in the real world robotics applications, we have tested our work on the task of reaching a goal in the workspace of the robotic arm, that task is a sub task of any industrial task for manipulators, in the following we will give the results of the experiment with an analysis and comments.

For the following hyperparameters:

- Number of the basis function of the controller = 50.
- Number of the iterations in each episode on the real robot (horizon) = 50.
- The control function is constrained to 0.1.

pan@isa.ru

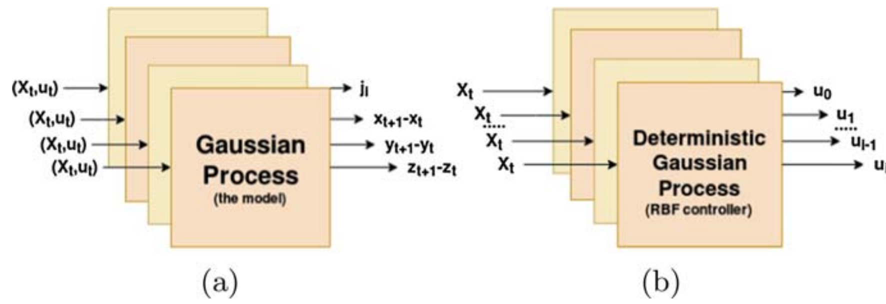

Fig. 7. (a) The transition model: multi-output Gaussian process regression model, the input is a for each sub GP is the state X_t and the control U_t, the output is the resulted difference of an output variable. (b) The controller: multi-output Gaussian process regression model, the input a for each sub GP is the state X_t, the output is a control variable u_i.

The average time results was:

- The time of interaction (running the robot) = 21.22 s.
- The time of training the transition model = 37.53 s.
- The time of optimizing the controller = 1380.95 s.
- The program running time = 1598 s

The corresponding plots for this case are presented on Fig. 8.

The algorithm can learn the inverse kinematics of the robot and achieved the task in a considerable time, and improved the trajectory also. Here we have to mention that, the performance of the algorithm was impressing because of the formulation of the experimental setup in a way exploiting the best of the PILCO algorithm, and matching the needs of such algorithm.

One of the interesting experimental results, is to monitor the confidence of our learned model, and how it match the real transition model. In the following we will list samples form one step prediction for the three coordinates of the end effector (Fig. 9).

5.2 Damaged Robot

To asses our implementation on one of the most interesting features of reinforcement learning algorithm, we have use a test case that could be happen for any robot. The damage of any joint's motor could lead the ordinary control algorithms to a complete fail in achieving the task. In our test case, we are considering the damage of a joint's motor (the joint which is surrounded by a red ellipse in Fig. 4), so the joint will be unresponsive.

We have used the same implementation with similar hyperparameters from the last experiment, the robot could adapt to the damage, and learned to achieve the task. The plots for that case in the right (Figs. 10 and 11). The speed of the learning process hasn't been affected by the damaging of the motor, because the algorithm doesn't depend on the dynamical model of the robot.

pan@isa.ru

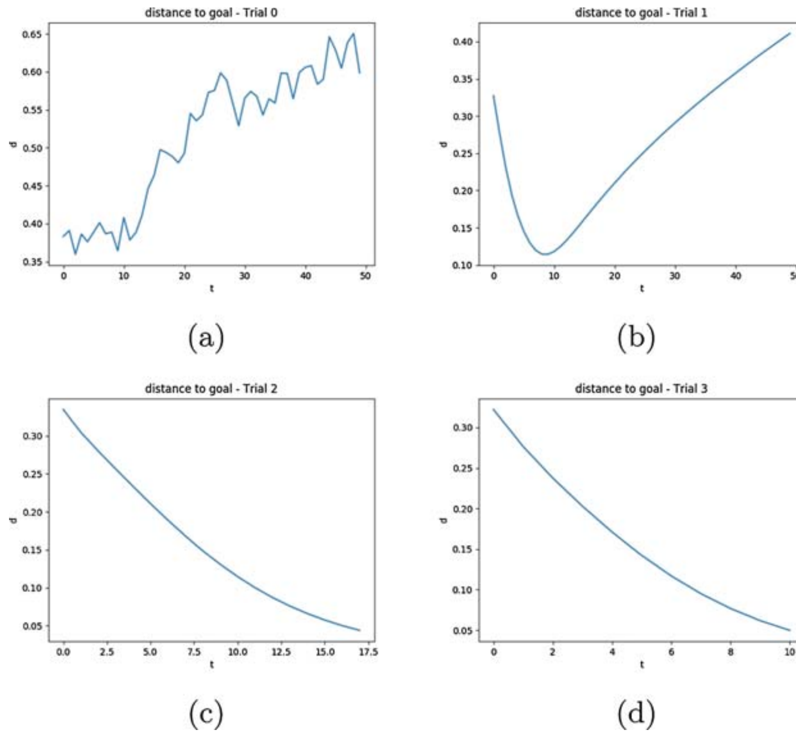

Fig. 8. The distance between the end effector and the target position, (a) is the random roll-out, (b) is the first roll-out after optimizing the controller in the simulation, we can see the robot approaches the target point here, but after that go away from it, but in (c) it learns to reach the goal after just 20 time steps, and improve that time by reach it in 10 time steps only in (d)

5.3 Comparison with the Matlab Implementation

We were interested in comparing how much using the computational graph could help us speed up the learning process for PILCO algorithm. We have tested our implementation in the Cartpole environment, with a similar hyperparameters to ones in the Matlab implementation, and same conditions.

The average time for running both implementations for 8 epochs:

– PILCO in python with computational graphs = 671 s.
– PILCO original implementation = 1265 s.

The using of the computational graphs, leverage the GPU power, and give as a speed up by a factor

$$S = \frac{T_{Matalb}}{T_{python}} = \frac{1265}{671} = 1.885.$$

Which is considered as a satisfying result.

pan@isa.ru

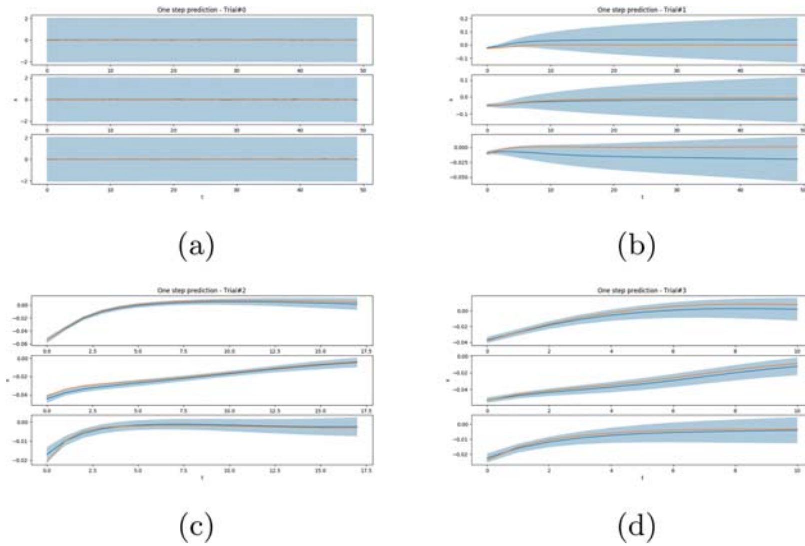

Fig. 9. The one step prediction of the transition model: the plot shows the difference between the states and the confidence level for each prediction; in (a) we can see the model is not confident in the first iteration. After the first learning epoch (b) it reduce the margin of uncertainty, but it failed to follow the real transition (orange line in the plot), the model has reduce the uncertainty about transition over the next iterations (c) and (d), reaching a small margin (d is better than c-check the scale of y-axis) (Color figure online)

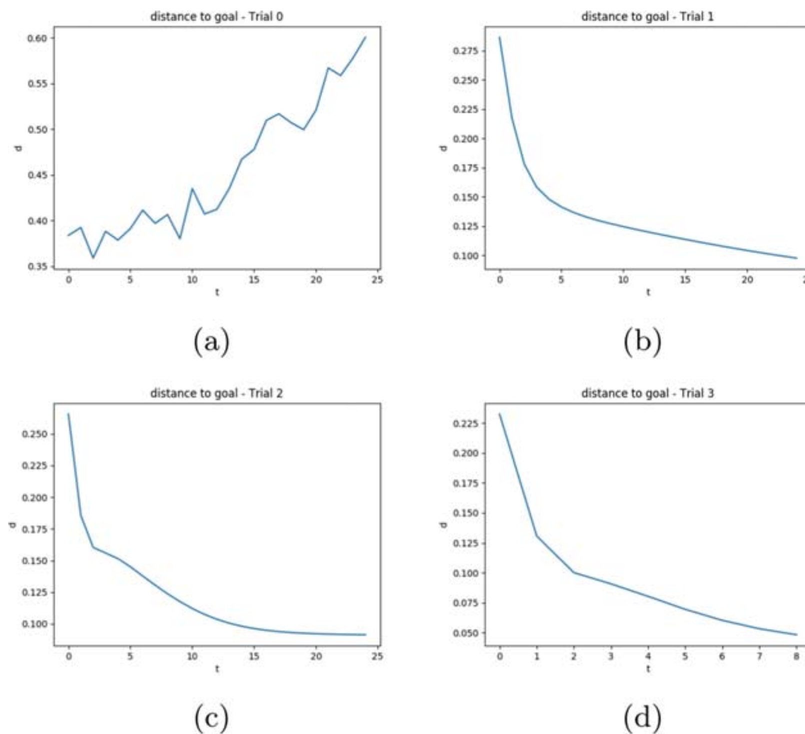

Fig. 10. The distance between the end effector and the target position - the case of an unresponsive joint, similar results to the previous case, the robot approaches the target in the second iteration (b) and third (c), reaches it in the in the fourth (d) after just 8 time steps.

pan@isa.ru

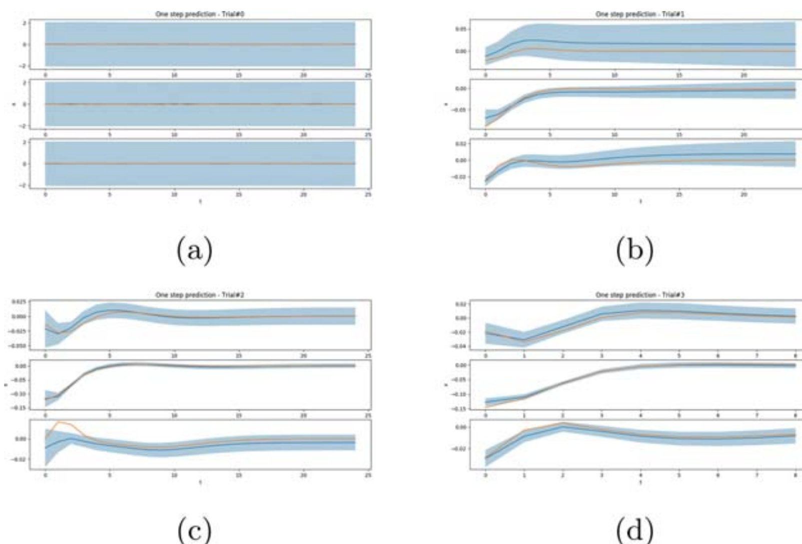

Fig. 11. The one step prediction of the transition model- the case of an unresponsive joint; confidence over the x axis was bigger than other two coordinates in (b), but the robot could learn to solve the task and reduce the margin of the uncertainty in (c) and (d).

6 Discussion and Future Work

Our implementation for PILCO in Python, with the using of computational graphs through GPflow gives the desired results in terms of the two goal proprieties (1) ROS friendly, where it is in python, and it is easy to describe a real world robotic experiment as a gym environment. (2) it leverage the computation power of the new hardware, so it could learn faster; while we couldn't feel the importance of this feature for our simple experiments, it may play an important role when working with more complex applications.

We have to mention to the compromise between long horizon and the accuracy of the model. Smaller horizon means less interaction on the real robot, which is desirable; but also mean smaller data set, and worse model for the state transition. So before deploying the learned controller, it is important to check the one step prediction plots over test (plausible for our task) trajectories. It is not recommended to deploy the controller after the first time achieving the task.

The importance of our work impedes in: (1) demonstrating the ability of adapt for robots when using algorithms that doesn't depends on the dynamical model (e.g reinforcement learning), (2) it is desirable to adapt as fast as possible, so it is important to work toward fast reinforcement learning, (3) using the probabilistic models and computing power could be the right method in achieving that goals.

We have many interesting points to work in this direction. While GP gives a good results, but it has some down points, like it can't handle discontinuities in the state, so using our implementation we can easily try to use Deep GPs instead of shallow GPs and compare the results. The computation complexity of the GP equals $O(n^3)$ which cause problem when working on a scale, in our

pan@isa.ru

work we used computational graphs to speed up the inference process of GPs, but we can work also on using GPU in other parts of the algorithm. In robotics, sometimes we are working with sparse rewards, so we can study how to solve such problems efficiently, and how we could make use of GPs.

References

1. McFarlane, D.C., Glover, K.: Robust Controller Design Procedure Using Normalized Coprime Factor Plant Descriptions. Lecture Notes in Control and Information Sciences, vol. 138. Springer, Heidelberg (1990). https://doi.org/10.1007/BFb0043199
2. Rocco, P.: Stability of PID control for industrial robot arms. IEEE Trans. Robot. Autom. **12**(4), 606–614 (1996)
3. Åström, K.J., Wittenmark, B.: Adaptive Control. Courier Corporation, Mineola (2013)
4. Wen, J.T., Murphy, S.H.: PID control for robot manipulators. Rensselaer Polytechnic Institute (1990)
5. Teixeira, R.A., Braga, A.D.P., De Menezes, B.R.: Control of a robotic manipulator using artificial neural networks with on-line adaptation. Neural Process. Lett. **12**(1), 19–31 (2000)
6. Nesnas, I.A., et al.: CLARAty: challenges and steps toward reusable robotic software. Int. J. Adv. Robot. Syst. **3**(1), 5 (2006)
7. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning, vol. 135. MIT Press, Cambridge (1998)
8. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. J. Artif. Intell. Res. **4**, 237–285 (1996)
9. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning, pp. 1889–1897, June 2015
10. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
11. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
12. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
13. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436 (2015)
14. Deisenroth, M.P., Neumann, G., Peters, J.: A survey on policy search for robotics. Found. Trends® Robot. **2**(1–2), 1–142 (2013)
15. Carlson, J., Murphy, R.R.: How UGVs physically fail in the field. IEEE Trans. Robot. **21**(3), 423–437 (2005)
16. Cully, A., Clune, J., Tarapore, D., Mouret, J.B.: Robots that can adapt like animals. Nature **521**(7553), 503 (2015)
17. Nagatani, K., et al.: Emergency response to the nuclear accident at the Fukushima Daiichi Nuclear Power Plants using mobile rescue robots. J. Field Robot. **30**(1), 44–63 (2013)
18. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. The MIT Press, Cambridge (2006)

pan@isa.ru

19. Ko, J., Klein, D.J., Fox, D., Haehnel, D.: Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In: Proceedings 2007 IEEE International Conference on Robotics and Automation, pp. 742–747. IEEE, April 2007
20. Wilson, A., Fern, A., Tadepalli, P.: Incorporating domain models into Bayesian optimization for RL. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010. LNCS (LNAI), vol. 6323, pp. 467–482. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15939-8_30
21. Engel, Y., Mannor, S., Meir, R.: Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In: Proceedings of the 20th International Conference on Machine Learning, ICML 2003, pp. 154–161 (2003)
22. Deisenroth, M.P., Fox, D., Rasmussen, C.E.: Gaussian processes for data-efficient learning in robotics and control. IEEE Trans. Pattern Anal. Mach. Intell. **37**(2), 408–423 (2015)
23. Matthews, D.G., et al.: GPflow: a Gaussian process library using TensorFlow. J. Mach. Learn. Res. **18**(1), 1299–1304 (2017)
24. Brockman, G., et al.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
25. http://www.ros.org
26. http://mlg.eng.cam.ac.uk/pilco/
27. http://www.incompleteideas.net/IncIdeas/BitterLesson.html

pan@isa.ru