Proceeding of the IEEE
International Conference on Robotics and Biomimetics
Dali, China, December 2019

Influences of Neural Network Structures on an Efficient Reinforcement Learning Policy Search

Wangshu Zhu and Andre Rosendo
School of Information Science and Technology
ShanghaiTech University
Shanghai, China
{*zhuwsh, arosendo*}*@shanghaitech.edu.cn*

***Abstract*— The Black-box Data-efficient RObot Policy Search algorithm, also known as Black-DROPS, is one of the most data-efficient algorithms for Reinforcement Learning for robotics. The algorithm is based on studying the dynamical model of uncertainty of robots, learning and optimizing the corresponding policy to maximize the reward. Black-DROPS does not limit the reward function, so it can have a wide range of applications. The algorithm is an optimization algorithm using the numerical approach, which is more efficient than the analytical approach in the case of multi-core operation. But the default one layer neural network does not have enough power to meet some difficult problems, and when faced with a high dimensional problem the algorithm will take an unusually long time to solve it. In this paper, we explore different Neural Network structures (layers, neurons, and computer cores) to study their influence on the speed from Black-DROPS on a Double Inverted Pendulum simulation. We demonstrate that although the performance (number of iterations to reach optimality) is affected by structural changes to a lesser extent, the computation time (time per iteration) is drastically affected by such changes, and an optimal structure size seems to exist, as we highlight in our results.**

***Index Terms*— Reinforcement Learning, Neural Network, Policy Search Algorithm**

I. Introduction

Robots in the real world will encounter unexpected situations, such as damage and malfunctions. A few learning-based algorithms [1], [2] or model-based algorithms [3] can help robots discover the environment, adapt their trajectories and continue to finish the tasks. Comparing to other learning algorithms (e.g., deep learning [4] which relies on huge data sets, e.g. 1.2 million labeled images in the ImageNet database [5]) reinforcement learning on robotics should be highly data-efficient. That means that the algorithm should be able to use finite data (interaction time between the robot and the world) to complete the tasks of the robot.

When robots learn in this scenario, where data is scarce, a common strategy is to extract as much data as possible from interactions, which means that every moment of the state needs to be taken into account. However, this idea is contrary to some current direct policy search algorithms (such as the policy gradient algorithm [6], Bayesian optimization algorithm [1]), which only uses the cumulative reward of each iteration. To use the continuous state of the robot, a better method is to learn the dynamical model of the robot [7] and then find a good policy according to the model. However, such a method is based on the better dynamical model. For real robots, there is often not enough interaction time to improve model quality through traditional methods.

To solve this problem we can take the uncertainty of the model into account in order to avoid overfitting the model [8], [9]. Following this strategy, the PILCO (Probabilistic Inference for learning COntrol) algorithm [8] learns a dynamical model with Gaussian processes (GPs) and uses a gradient-based optimizer to search for a policy that maximizes the expected long-term reward with the uncertainty of the model. While, it also presents some defects: the constraints on the reward functions and policies, and the computing process cannot be parallelized with multiple cores.

The Black-DROPS algorithm [9] is another model-based policy search algorithm for robotics which uses black-box optimizers CMA-ES (Covariance Matrix Adaptation Evolutionary Strategies) [11] and can take advantage of parallel computing to a great extent (up to 6x speed-up and 1.6x faster than PILCO when 12 cores are used). It uses the Monte Carlo method and a distribution-based black-box optimization algorithm CMA-ES [11]. In addition, it removes the constraints of PILCO and can ensure the computing efficiency because the CMA-ES used by the algorithm is based on the ranking algorithm, so only the best sampling points of each generation of sampling points will be calculated.

While the Black-DROPS algorithm can speed up the computing when working with multiple cores, here comes the difficulties when the model-based policy meet with some complex problems: as the algorithm models the transition function between full state or action spaces (joint positions, joint velocities, etc.), the complexity of the model increases substantially with each new degree of freedom. Unfortunately, the quantity of data required to learn a good model scales most of the time exponentially with the dimension of the state space [12]. As a result, the data-efficiency of model-

*This work was supported by the National Natural Science Foundation of China, project number 61850410527, and the Shanghai Young Oriental Scholars project number 0830000081.

based approaches suffers considerably from the increase of the dimensionality of the model.

One way of handling "the curse of dimensionality" is to use different structures of neural networks used in the policy to accelerate the learning speed. The neural network has succeeded in a lot of aspects in reinforcement learning [13], [14], [15]. It is quite powerful with different numbers of layers and numbers of neurons per layer to do segmentation, regression, classification, etc. The ideal model-based policy search algorithm with multiple layers neural networks should:

(1) be able to handle high dimensional and complex robots (e.g., multiple inverted pendulum or soft robots);

(2) still consider the uncertainty of the dynamical model during policy search;

(3) remove constraints on the reward function;

(4) remove constraints on the policy representation;

(5) be able to implement parallel computing to accelerate computation.

We demonstrate these features on a classic control problem: the double inverted pendulum with three families of policies, neural networks, and Gaussian processes and linear network in simulation. We show that our approach is able to learn policies in about 30 seconds of interaction time to control the double pendulum swing up to the inverted position, which is about 30% faster than the original Black-DROPS algorithm.

II. Related work

Reinforcement learning [13] is a special kind of machine learning algorithm compared to supervised learning and unsupervised learning. The algorithm is based on the pre-existing environmental state to determine an action to execute on the robot. After the interaction with environment the robot will obtain a reward value and a new state. Then, the algorithm will enter back to the loop. The ultimate goal is to make the (cumulative) reward maximized. Many games such as Go [13], Chess [16], and Atari games [17] can be solved by different reinforcement learning methods. Reinforcement learning will determine the action by continually trying and training to obtain the corresponding policy.

For robotics, direct policy search has been widely applied to the reinforcement learning problem of continuous state actions from low-dimensional to high-dimensional. Early reinforcement learning explored the action space using gradient methods and probabilistic strategies. But one of its major drawbacks is that the multivariate prediction gradient will converge slowly. The parameter-based policy gradient exploration algorithm (PGPE) [18] solves this problem to some extent by transforming the exploration into parameter space. The PoWER algorithm [19] learn policies based on weighted exploration with returns algorithm uses an average of probability weights, which conforms to natural gradients but does not calculate gradients. But the algorithm assumes that the sum of immediate reward is a positive number, so the design of the reward functions can be complicated. The use of path and strategy promotion algorithm (PI^2) [20] does not have the assumption. But when the two reward functions are the same, but the final effect of PoWER and PI^2 are the same.

One limitation of the PGPE algorithm is that it only considers the parameters in a separated point of view instead of taking correlation of the parameter space into account, which can be considered to solve by Natural Evolution Strategies (NES) [21] and Covariance Matrix Adaptation Evolutionary Strategies (CMA-ES) [11]. Both of these methods are black-box optimizers (BBOs), and the distribution of a search is iteratively updated by calculating the average of the parameters and the estimated gradient of the variance. In each iteration, the Evolutionary Strategies algorithm samples some vectors, sorts the sample vectors, and iterates to obtain the next generation by the best samples.

In general, black-box optimization can be applied in direct policy search, but it is necessary to pay attention to two problems of efficiency: the efficiency of model optimization, and the efficiency of robot interaction. For the former, Bayesian optimization [22] one of the black-box optimizations can be used to improve efficiency. Bayesian optimization is a special black-box optimization problem. It accelerates learning by establishing an alternative model of the target equation. The efficiency of the optimization model can be improved by the balance between exploration and development. The interaction efficiency between the robot and the environment can be improved by model-based algorithms, such as the model-based PGPE algorithm and the PILCO algorithm using gradient-based strategy updates.

It has been proved that Black-DROPS [9] and PILCO [8] are two of the most data-efficient model-based policy search algorithms for robotics. But they are different from each other in how they use the uncertainty of the model and how they optimize the policy given the model: PILCO uses moment matching and analytical gradients, while Black-DROPS uses Monte-Carlo rollouts and a black-box optimizer the CMA-ES [11]. Compared with PILCO, Black-DROPS has two main superiorities: (1) any reward function or policy can be implemented because the algorithm imposes no constrains on that, and (2) it can take advantage of multi-core computers by parallel computing. Due to these advantages, Black-DROPS can find the optimal policy faster than PILCO in standard control benchmarks (inverted pendulum and cart-pole swing-up) and achieves similar data-efficiency to PILCO.

Model-based policy search algorithms reduce the required interaction time, but for more complex or higher dimensional systems, they still require even hundreds of iterations to find a working policy. One way to reduce the interaction time without leaning models is by using different structures of neural networks in the policy parameterization.

III. Problem formulation

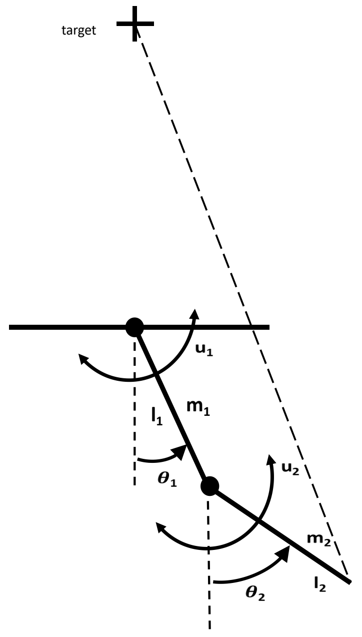

Fig. 1. The double inverted pendulum system where there are two actuators applying torques $u_1 and u_2$ on the link points. The initial position is the natural drooping position while the target position is the fully inverted state.

In this article, we consider dynamical systems of the form:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + f(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{w} \tag{1}$$

with continuous-valued state $\mathbf{x} \in \mathbb{R}^E$ and controls $\mathbf{u} \in \mathbb{R}^F$, i.i.d. Gaussian system noise $\mathbf{w} \sim \mathcal{N}(0, \sum_w)$, and unknown transition dynamics f. The policy search objective is to find a deterministic policy π, $u = \pi(\mathbf{x}|\boldsymbol{\theta})$, which maximizes the expected long-term reward when following policy π for T time steps:

$$J(\boldsymbol{\theta}) = \mathbb{E}[\sum_{t=1}^{T} r(\mathbf{x}_t)] \tag{2}$$

where $r(\mathbf{x}_t)$ is the step reward or called immediate reward of being in state $\mathbf{x}$ at time t. We assume that π is a function parameterized by $\boldsymbol{\theta} \in \mathbb{R}^\theta$ and the step reward function $r(\mathbf{x}) \in \mathbb{R}$ might be unknown to the learning algorithm.

The double inverted pendulum system is a two-link robot arm with two actuators (Fig. 1). Each of the link is of length 0.5 m and mass 0.5 kg so $l_1 = l_2 = 0.5$ and $m_1 = m_2 = 0.5$. The angles of the links are marked as θ_1 and θ_2 while the angular velocities are $\dot{\theta_1}$ and $\dot{\theta_2}$. Both of the torques u_1 and u_2 are constrained to [-3,+3] Nm. The control signal is sent by every 0.1s. The objective is to learn a controller to apply some torques during each interaction time and swing the double inverted pendulum up to the target position and keep it in balance.

IV. Methods

A. Model Learning

To extract as much information from finite interaction time, we need to use Gaussian processes to find a model $\hat{f}$ that approximates the unknown dynamics f of our systems with uncertainty information. A Gaussian process (GP) is an extension of the multivariate Gaussian distribution to an infinite-dimension stochastic process for which any finite combination of dimensions will also be a Gaussian distribution [23].

We use tuples composed of the state vector $\mathbf{x}_t$ and the action vector $\mathbf{u}_t$ which is $\widetilde{\mathbf{x}_t} = (\mathbf{x}_t, \mathbf{u}_t) \in \mathbb{R}^{E+F}$ as inputs. We use the difference between the current state vector and the next one: $\Delta\mathbf{x}_t = \mathbf{x}_{t+1} - \mathbf{x}_t \in \mathbb{R}^E$ as training targets. And using E independent Gaussian processes we can model each dimension of the difference vector $\Delta\mathbf{x}_t$. For each dimension d= 1,..., E of $\Delta\mathbf{x}_t$, the GP is computed as:

$$\hat{f}_d(\widetilde{x}) \sim \mathcal{GP}(\mu(\widetilde{\mathbf{x}}), k(\widetilde{\mathbf{x}}, \widetilde{\mathbf{x}}')) \tag{3}$$

where k is the kernel function) and μ is the mean function. Assuming there is a set of observations $D^d_{1:t} = [f_d(\mathbf{x}_1), ..., f_d(\mathbf{x}_t)]$, we can inquire the GP at a new input point $\widetilde{\mathbf{x}}_{new}$ about the probability:

$$p(\hat{f}_d(\widetilde{\mathbf{x}}_{new})|D^d_{1:t}, \widetilde{\mathbf{x}}_{new}) = \mathcal{N}(\mu(\widetilde{\mathbf{x}}_{new}), \sigma^2(\widetilde{\mathbf{x}}_{new})) \tag{4}$$

A kernel vector $k = k(D^d_{1:t}, \widetilde{\mathbf{x}}_{new})$ and a kernel matrix K are used to compute the mean and variance predictions of this GP [9]. Same with the Black-DROPS algorithm we use the exponential kernel with automatic relevance determination [9]:

$$k(\widetilde{\mathbf{x}}_p, \widetilde{\mathbf{x}}_q) = \sigma^2 exp(-\frac{1}{2}(\mathbf{x}_p - \mathbf{x}_q)^T \Lambda^{-1}(\mathbf{x}_p - \mathbf{x}_q)) + \delta\sigma^2_{n_d} \tag{5}$$

where δ equal to 1 when p = q and 0 otherwise. The $\Lambda, \sigma^2, \sigma^2_{n_d}$ are the hyper-parameters of the kernel found by the Maximum Likelihood Estimation [10].

We can also use a GP to learn the step reward function which can gain a reward $r(\mathbf{x}) \in \mathbb{R}$ according to each state $\mathbf{x}$:

$$\hat{r}_d(x) \sim \mathcal{GP}(\mu_r(\mathbf{x}), k_r(\mathbf{x}, \mathbf{x}')) \tag{6}$$

which is similar to Eq. 4 by GP.

B. Policy Evaluating

The objective is to find an optimal policy to maximize the expected cumulative reward Eq. 2 which requires predicting the state evolution according to an uncertain transition model and an uncertain reward model. Assuming that these two models have been learned well in the model learning section, we can predict the state evolution accurately.

Specifically, Black-DROPS uses a Monte Carlo approximation of the distribution of the state: at each step, it samples a state by the GP of the model and samples its reward by the reward model, inquires the policy to choose the action and applies the action on the state. Then, the algorithm will get a new state and use this new state to sample the next state. By keep performing this process, we will finally obtain a relatively optimal estimate of the expected cumulative reward [24]. After predicting the state evolution, we can accumulate all step rewards of the state to evaluate the policy as $J(\boldsymbol{\theta})$.

C. Policy Searching

Then we can use CMA-ES (Covariance Matrix Adaptation Evolutionary Strategies) [11] to do the policy searching. The CMA-ES is a population-based Black-Box optimizer and it is composed of four steps at the k_{th} iteration .
(1) sample n new candidates by a multivariate Gaussian distribution of mean m_k and covariance $\sigma_k^2 C_k$:

$$\boldsymbol{\theta_i} \sim \mathcal{N}(m_k, \sigma_k^2 C_k) \tag{7}$$

for i ∈ 1, 2 ,..., n
(2) rank the n samples based on their expected accumulative reward $J(\boldsymbol{\theta})$
(3) compute m_{k+1} by averaging the μ best candidates:

$$m_{k+1} = \frac{1}{\mu}\sum_{i=1}^{\mu} \boldsymbol{\theta_i} \tag{8}$$

(4) update the covariance matrix $\sigma_k^2 C_k$ to reflect the new distribution of the μ best candidates. Then come back to the loop until the candidates converge which means we find a local optimal policy at least.

D. Policy Representations

We mainly use three policy representations in this paper: Neural Network, Linear Network, and the Gaussian process policy.

The network equation of the i_{th} layer is computed by $\mathbf{y}_i = \phi(\mathbf{W}_i\mathbf{y}_{i-1}+\mathbf{b}_i)$ and $\mathbf{y}_0 = \phi(\mathbf{W}_0\mathbf{x}+\mathbf{b}_0)$, where $\mathbf{W}_i$ is the weight matrix ,$\mathbf{b}_i$ is the bias vector and ϕ is the activation function which is tanh function in the paper. $\mathbf{y}_{i-1}$ and $\mathbf{y}_i$ are the input and output vector. We need to find the optimal policy $\phi^*(\mathbf{x})$ by the $MAX(\mathbb{E}[\sum_{t=1}^{T} r(\mathbf{y}_{n_t})])$, where n is the number of layers of the neural network. By default, the layer of the neural network is 1 and the number of neurons per layer is 10.

The Linear Network is quite similar to the Neural Network which just removes the activation function. This means the multi-layer Linear Network can be simplified to one layer Linear Network by some matrices multiplication. So, the expandability of Linear Network is relatively weak and the power of representation of Linear Network is not as strong as Neural Network as a result.

The Gaussian process policy is defined by the covariance matrix K and the kernel **k** like in Black-DROPS [9].

E. Algorithm gathered

Put all methods together, we can get the skeleton of the model-based policy search.

Algorithm 1 Model-based policy search

```
1: Initialize the policy $\pi$ with random parameters $\theta$
2: iteration=0
3: while Task is not solved or iteration == limitation do
4:     Execute policy on the robot and record data
5:     Learn the dynamical model from the gathered data
6:     Learn the step reward function r from the gathered data
7:     Optimize $\theta$ according to J($\theta$) using CMA-ES
8:     iteration+=1
9: end while
```

F. Reinforcement Learning

The double inverted pendulum simulation has the following elements: State, Action and Reward in reinforcement learning.
(1) **State**: $\mathbf{x} = [\dot{\theta_1}, \theta_1, \dot{\theta_2}, \theta_2] \in \mathbb{R}^4$, while initial state $\mathbf{x}_0$ = [0,0,0,0].
To avoid the angular difference like 2π and 4π, we need to transform the input $\mathbf{x}$ to $\mathbf{x}_{transform} = [\dot{\theta_1}, cos(\theta_1), sin(\theta_1), \dot{\theta_2}, cos(\theta_2), sin(\theta_2)] \in \mathbb{R}^6$
(2) **Action**: $\mathbf{u}$ =[u_1,u_2] $\in \mathbb{R}^2$, $-3 \leq u_i \leq 3N$.
(3) **Reward**: $r(\mathbf{x}) = exp(-\frac{1}{2{\sigma_c}^2}(\mathbf{x}-\mathbf{x}_*)^T\mathbf{Q}(\mathbf{x}-\mathbf{x}_*))$ which is a negative correlating distance-based reward function where ${\sigma_c}^2$ is the variance, $\mathbf{x}_*$ is the target state and r($\mathbf{x}$)∈[0,1], $\mathbf{Q}$ is a weight matrix and we can set some entries to 0 to ignore some influence of the corresponding entries of $\mathbf{x}$ on the reward function. By default we will ignore influence of the angular velocity on the reward function.

V. Results and Discussion

A. Influences of Policy-search methods

We first apply three different policy-search methods: Neural Network, Linear Network and Gaussian process on the same double inverted pendulum simulation. As it can be seen in Fig. 2, the NN policy and LN policy have a relatively faster convergence (finding the optimal policy at the 10th iteration) and can achieve higher rewards than the GP policy in the same iteration. This is probably because the dimension of the parameters is high for the GP which will make it easily overfit. The NN policy can have a better performance with an activation function in the long term (after about 10 iterations) than LN did.

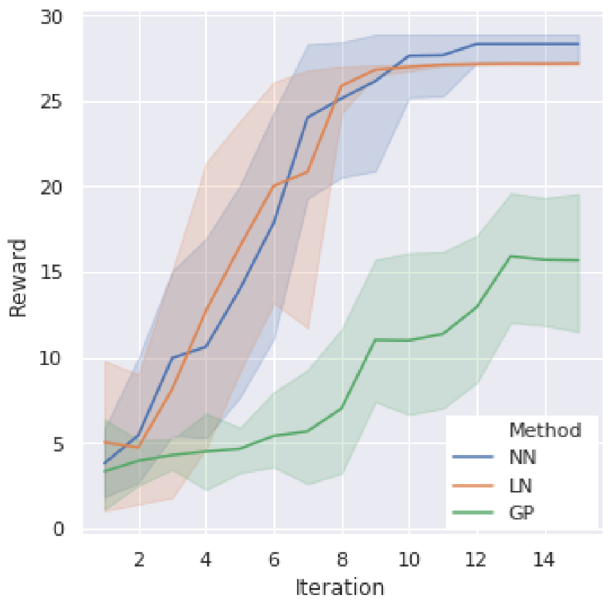

Fig. 2. Result for the double inverted pendulum task with different methods (30 replicates of each method). The lines are median values and the shaded regions 95% confidence interval. The NN method and LN method converges to higher quality solution in fewer iterations than GP policy.

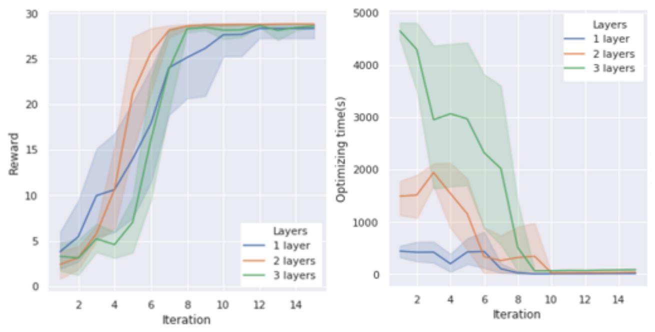

Fig. 3. Results for the double inverted pendulum task with different numbers of layers with NN method (30 replicates of each method). The lines are median values and the shaded regions 95% confidence interval. The figure on the left shows that the NN with 2 layers converges to higher quality solution in fewer iterations than the NN with 1 or 3 layers. And in general, the NN with more complex structure will take more optimizing time in each iteration.

B. Influences of Layers

The NN policy has the best performance within three policies. We change the original 1 layer of NN in [9] to 2 and 3 to identify the influences of the layers of NN on the reinforcement learning policy search. In our simulations we found that the NN with 2 layers can find the optimal policy at about 7th iteration, which is faster than the 1 layer as Fig. 3. The 3 layers NN is also better than the original 1 layer but it does not mean that more layers generate a better performance. In reverse, the 3 layers NN converges slower than the 2 layers NN, which can be caused by the overfitting of GP with too many parameters. We also analyze the optimizing time of NN with different layers, as seen in the right side of Fig. 3. Intuitively, the NN with more layers will have more parameters causing the dimension of the optimization higher. As a result, we need additional time to optimize the policy.

C. Influences of Neurons

We also change the number of neurons per layer from 10 as [9] to 15 or 20 to identify the influence on the policy search as Fig. 4. The figure on the left shows that more neurons per layer will not improve the performance (reward by iteration) but increase the optimizing time considerably as the parameters grows as the right one shows. The probable reason is the overfitting of GP with too many parameters. In addition, the NN with 5 neurons per layer will not find a good policy in general. The 5 neurons NN may not be powerful enough to solve this problem with a high dimension.

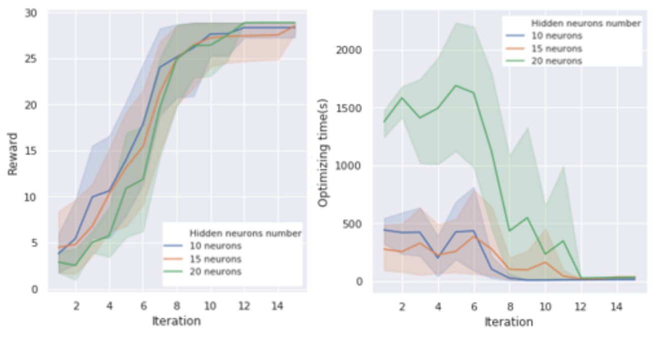

Fig. 4. Results for the double inverted pendulum task with different neurons per layer of NN (30 replicates of each method).The lines are median values and the shaded regions 95% confidence interval. The NN with more neurons will not perform better. And the NN with more complex structures will take more optimizing time in each iteration.

D. Influences of Cores

We test the benefits of parallel computing on our task with one layer NN method and show that it can reach up to 2x speedup when 12 cores are used which accurately is smaller than the 6x speed increase mentioned in [9] (Fig. 5). The probable reason is that the dimension of the problem has exceeded the ceiling of parallel computing of Monte Carlo approaches or CMA-ES.

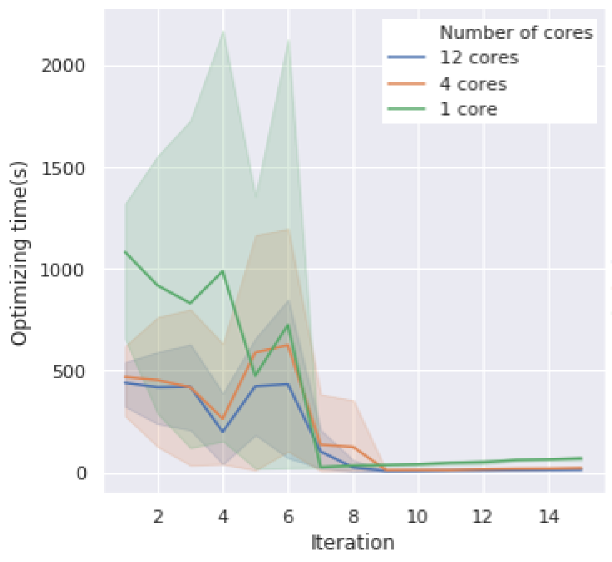

Fig. 5. Speeding up for the double inverted pendulum task with different numbers of cores with the same one layer NN method (30 replicates of each method). The lines are median values and the shaded regions 95% confidence interval. As more cores are being used, our algorithm benefits from it and has up to 2x speed-up when 12 cores are used.

VI. Conclusion

Different structures of neural networks have a big impact on the reinforcement learning policy-search, which can improve the speed of convergence and the reward per iteration. But with more complex structures, such as more layers or more neurons per layer, we will probably spend more computational time in general. When we are applying NN for reinforcement learning policy-search algorithms, we should, considering the complexity of the model (dimensionality) that we are focusing on, consider the trade-off between the performance and computational time.

Another point is that Gaussian processes have a great probability of overfitting when the number of parameters is relatively large, which should be cautiously considered in the policy search algorithm. We can adopt parallel computing to speed up the optimization but the room for that depends on the complexity of the problem (mostly the dimension space).

In the future, we will apply the model-based policy search algorithm with different structures on more complicated problems and real robots. We consider taking into account environment properties for the real robots , as simulation environments are ideal and real world applications should consider noises, disturbances and uncertainties from motors and sensors.

References

[1] A. Cully, J. Clune, D. Tarapore, JB. Mouret. Robots that can adapt like animals. Nature. 2015 May;521(7553):503.

[2] K. Chatzilygeroudis , V. Vassiliades, JB. Mouret. Reset-free trial-and-error learning for robot damage recovery. Robotics and Autonomous Systems. 2018 Feb 1;100:236-50.

[3] Y. Zhang, W. Zhu, A. Rosendo. QR Code-Based Self-Calibration for a Fault-Tolerant Industrial Robot Arm. IEEE Access. 2019 Jun 3;7:73349-56.

[4] Y. LeCun ,Y. Bengio , G. Hinton. Deep learning. nature. 2015 May;521(7553):436.

[5] J. Deng, W. Dong, R. Socher , LJ. Li, K. Li, L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In2009 IEEE conference on computer vision and pattern recognition 2009 Jun 20 (pp. 248-255). Ieee.

[6] MP. Deisenroth, G. Neumann , J. Peters. A survey on policy search for robotics. Foundations and Trends in Robotics. 2013 Aug 30;2(12):1-42.

[7] D. Nguyen-Tuong, J. Peters. Model learning for robot control: a survey. Cognitive processing. 2011 Nov 1;12(4):319-40.

[8] MP. Deisenroth, D.Fox, CE. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. IEEE transactions on pattern analysis and machine intelligence. 2013 Nov 4;37(2):408-23.

[9] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, JB. Mouret. Black-box data-efficient policy search for robotics. In2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2017 Sep 24 (pp. 51-58). IEEE.

[10] CE. Rasmussen. Gaussian processes in machine learning. InSummer School on Machine Learning 2003 Feb 2 (pp. 63-71). Springer, Berlin, Heidelberg.

[11] N. Hansen, A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. Evolutionary computation. 2001 Jun;9(2):159-95.

[12] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, Q. Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. International Journal of Automation and Computing. 2017 Oct 1;14(5):503-19.

[13] D. Silver, A. Huang, CJ. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman. Mastering the game of Go with deep neural networks and tree search. nature. 2016 Jan;529(7587):484.

[14] S. Levine, V. Koltun. Guided policy search. InInternational Conference on Machine Learning 2013 Feb 13 (pp. 1-9).

[15] B. Zoph, QV. Le. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578. 2016 Nov 5.

[16] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science. 2018 Dec 7;362(6419):1140-4.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, AA. Rusu, J. Veness, MG. Bellemare, A. Graves, M. Riedmiller, AK. Fidjeland, G. Ostrovski, S. Petersen. Human-level control through deep reinforcement learning. Nature. 2015 Feb;518(7540):529.

[18] F. Sehnke, C. Osendorfer, T. Rckstie, A. Graves, J. Peters, J. Schmidhuber. Policy gradients with parameter-based exploration for control. InInternational Conference on Artificial Neural Networks 2008 Sep 3 (pp. 387-396). Springer, Berlin, Heidelberg.

[19] J. Kober, J. Peters. Policy search for motor primitives in robotics. InLearning Motor Skills 2014 (pp. 83-117). Springer, Cham.

[20] E. Theodorou, J. Buchli, S. Schaal. A generalized path integral control approach to reinforcement learning. journal of machine learning research. 2010;11(Nov):3137-81.

[21] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, J. Schmidhuber. Natural evolution strategies. The Journal of Machine Learning Research. 2014 Jan 1;15(1):949-80.

[22] B. Shahriari, K. Swersky, Z. Wang, RP. Adams, N. De Freitas. Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE. 2015 Dec 10;104(1):148-75.

[23] CE. Rasmussen, H. Nickisch. Gaussian processes for machine learning (GPML) toolbox. Journal of machine learning research. 2010;11(Nov):3011-5.

[24] A. Kupcsik, MP. Deisenroth, J. Peters, AP. Loh, P. Vadakkepat, G. Neumann. Model-based contextual policy search for data-efficient generalization of robot skills. Artificial Intelligence. 2017 Jun 1;247:415-39.