Synthesizing Neural Network Controllers with Probabilistic Model-Based Reinforcement Learning

Juan Camilo Gamboa Higuera, David Meger and Gregory Dudek
School of Computer Science
Centre for Intelligent Machines
McGill University
{gamboa,dmeger,dudek}@cim.mcgill.ca

1 Introduction

Model-based reinforcement learning (RL) is an attractive framework for addressing the synthesis of controllers for mobile robots, due to its promise of data-efficiency. By using experience data (empirical data from real robotic systems) to fit state-transition dynamics models, an RL agent can search for good controllers using these models to evaluate policies in simulation; minimizing costly trials on the target robot platform. Minimizing interactions, however, means that experience datasets won't be large enough to obtain accurate dynamics models. Bayesian models are very helpful in this situation. Instead of requiring an accurate model, the RL agent may keep track of a distribution of dynamics models that are compatible with its experience. Evaluating a controller now involves quantifying its performance over this distribution: a good controller should perform well on any dynamics model drawn from the model distribution, to improve its chances of working in the real world. Two successful applications of this idea are the PILCO and Deep-PILCO algorithms.

PILCO [1] uses Gaussian Process (GP) models to fit one-step dynamics and networks of radial basis functions (RBFs) for feedback policies. PILCO has been shown to perform very well with little data in benchmark simulated tasks and on real robots [1]. We have used PILCO successfully for synthesizing swimming controllers for an underwater swimming robot [2]. However, PILCO is computationally expensive. Model fitting scales cubically with dataset size and simulated rollouts scale quadratically with state dimensionality, limiting its applicability to scenarios with small datasets and a low number of dimensions. Deep-PILCO [3] aims to address these shortcomings by employing Bayesian Neural Networks (BNN), implemented using binary dropout [4, 5]. Deep-PILCO uses BNNs to fit a dynamics model, and performs a sampling-based procedure for simulation. This Policy Search and model learning are done via stochastic gradient optimization: computations with linear scaling in the dataset size and state dimensionality. Deep-PILCO has been shown to result in better performing policies for the cart-pole swing-up benchmark task, but reduced data efficiency when compared with PILCO on the same task.

In this work, we extend on the results of [3]. We demonstrate that Deep-PILCO can match the data efficiency of PILCO on the cart-pole swing-up task with a small number of heuristics to stabilize optimization. We demonstrate that using neural network controllers improves the data efficiency of Deep-PILCO. We also show that using Log-Normal multiplicative noise [6] removes the need for searching for the appropriate dropout rate, and outperforms the original version of Deep-PILCO with binary dropout on the cart-double pendulum swing-up task.

2 Methods

We follow the problem definition in [1, 3]. We assume target systems that can be modeled with an unknown discrete-time first-order dynamics equation $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$, where f is unknown, $\mathbf{x}_t \in \mathbb{R}^D$ is the state of the system and $\mathbf{u}_t \in \mathbb{R}^U$ is the applied control, at time step t. The goal

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

is to find the parameters θ of a control policy π_θ that minimize a state-dependent cost function c accumulated over a given time horizon H,

$$\underset{\theta}{\arg\min} \quad J(\theta) = \mathbb{E}_\tau \left\{ \sum_{i=1}^{H} c(\mathbf{x}_t) \right\}. \tag{1}$$

The expectation is taken over the trajectory distribution $p(\tau) = p(\mathbf{x}_1, \mathbf{u}_1, ..., \mathbf{x}_H, \mathbf{u}_H)$. In PILCO, the expectation over trajectories is replaced by an expectation over individual state distributions $p(\mathbf{x}_1), ..., p(\mathbf{x}_H)$ which are modelled as Gaussian. Simulation consists of finding the parameters of each $p(\mathbf{x}_t)$, its mean and covariance, by analytical integration. The cost function is approximated as $J(\theta) \approx \sum_{i=1}^{H} \mathbb{E}_{\mathbf{x}_t} \{c(\mathbf{x}_t)\}$. This is an approximation to $p(\tau)$ as a multivariate Gaussian with block-diagonal covariance, assuming deterministic controllers.

Deep-PILCO uses the same approximation for the cost, but the state distributions $p(\mathbf{x}_t)$ are obtained empirically from simulated trajectories. We obtain K simulated trajectories from the following model:

$$\begin{aligned} &\mathbf{x}_1^{(k)} \sim p(\mathbf{x}_1), && f^{(k)} \sim p(f), \\ &\mathbf{u}_t^{(k)} \sim \pi_\theta \left(\mathbf{u}_t^{(k)} | \mathbf{x}_t^{(k)} \right), && \mathbf{y}_{t+1}^{(k)} \sim p(\mathbf{x}_{t+1}^{(k)} | \mathbf{x}_t^{(k)}, \mathbf{u}_t^{(k)}, \hat{f}^{(k)}), \\ & && \mathbf{x}_{t+1}^{(k)} \sim N(\mathbf{x}_{t+1}^{(k)} | \hat{\mu_{\mathbf{y}_t}}, \hat{\Sigma_{\mathbf{y}_t}}). \end{aligned} \tag{2}$$

The sampling from the dynamics model distribution $p(f)$ is approximated by sampling the weights of a BNN fit to the experience data. The last step in Eq. 2 corresponds to sampling from the moment-matched Gaussian distribution of the output of the dynamics.

2.1 Stabilizing the optimization procedure

2.1.1 Common random numbers (CRN) and PEGASUS policy evaluation

The convergence of Deep-PILCO is highly dependent on the variance of the estimated gradient $\nabla_\theta \hat{J}(\theta) = \frac{1}{K} \sum_k^K \nabla_\theta J(\theta | \hat{f}^{(k)}, \mathbf{x}_1^{(k)})$. The variance of this gradient is, among other things, dependent on $p(\mathbf{x}_1)$, the random numbers used for Monte Carlo integration, and the number of particles K. In our initial experiments we increased K from 10 to 100, and found it to improve convergence with small penalty on running time.

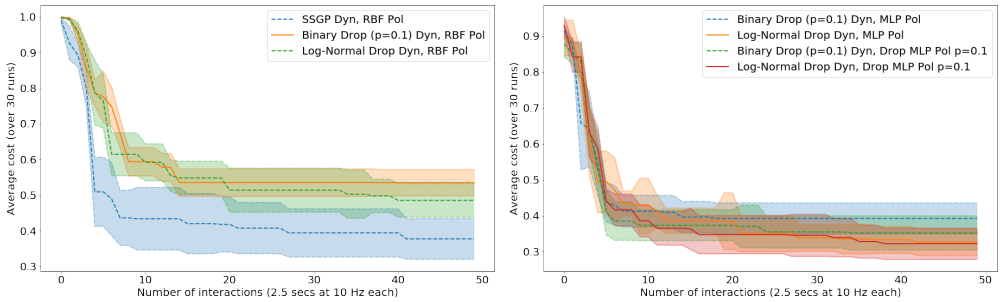
Increasing the number of particles also enables us to further reduce variance by using *common random numbers* (CRN). The generative model in Eq. 2, implemented with BNNs in Deep-PILCO, relies on the reparametrization trick to propagate policy gradients over the simulation horizon. If we provide the random numbers for the policy evaluation step of Deep-PILCO, we obtain a variant of the PEGASUS algorithm [7], which requires intermediate moment-matching and resampling steps.

2.1.2 Gradient Clipping

As noted in [3], the recurrent application of BNNs to implement the generative model in Eq. 2 can be interpreted as a Recurrent Neural Network (RNN) model. As such, Deep-PILCO is prone to suffer from vanishing and exploding gradients [8], especially when dealing with tasks that require long time horizon or very deep models for the dynamics and policy. Although numerous techniques have been proposed in the RNN literature, we found the gradient clipping strategy to be effective in our experiments; as shown in Figure 2b.

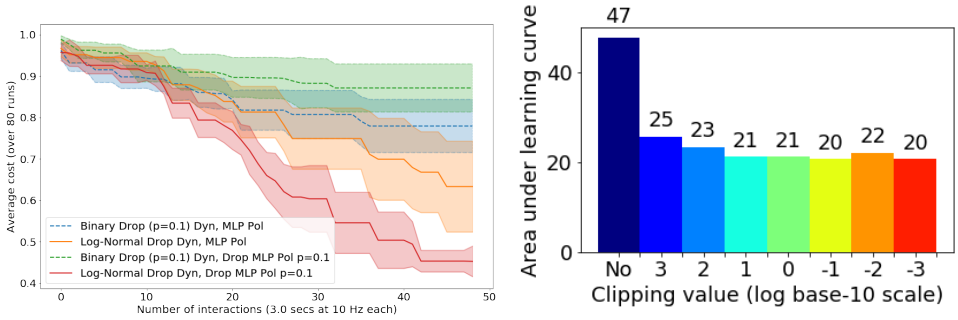
2.2 Training neural network controllers

While Deep-PILCO had been limited to training single-layer Radial Basis Function policies, the application of gradient clipping and CRNs allows stable training of deep neural network policies, opening the door for richer behaviors. We found that adding dropout regularization to the policy networks improves performance. During policy evaluation, we sample from policies with a similar scheme to that used for dynamics models. For each particle, we sample one dynamics model and one policy, and keep them fixed during policy evaluation. This can be viewed as attempting to learn a



(a) Cartpole RBF Policies (b) Cartpole Deep Policies

Figure 1: Cost per trial on the cart-pole swing-up task. The shaded regions correspond to half a standard deviation (for clarity). In (a), RBF policies are learned with various methods, as has been examined by prior work. NN policies are learned in (b), which demonstrate that Deep-PILCO, with deterministic policy evaluations, can match the data efficiency of PILCO in the cartpole task.



(a) Double CP Learning (b) Clipping comparison

Figure 2: (a) Cost per trial on the double cart-pole swing-up task. The shaded regions correspond to half a standard deviation (for clarity). This demonstrates the benefit of using Log-Normal multiplicative noise for the dynamics with dropout regularization for the policies (b) Shows the area under the learning curve for the cart-pole (sum of average cost across all trials – lower is better) for various gradient clipping strategies.

distribution of controllers that are likely to perform well over plausible dynamic models. We decided to make the policy stochastic during execution, by using a single dropout sample of the policy at every time step. This provides a limited amount of exploration, which we found beneficial, in particular, for the double cart-pole swing-up task.

2.3 BNNs with Log-Normal multiplicative noise

Deep-PILCO with binary dropout requires tuning the dropout probability to a value appropriate for the model size. We experimented with other BNN models [9, 10, 6] to enable learning the dropout probabilities from data. The best performing method in our experiments was using Log-Normal dropout with a truncated Log-Uniform prior $\text{LogU}_{[-10,0]}$; i.e. the multiplicative noise is constrained to values between 0 and 1 [6]. Other methods we may explore in the future include Concrete Dropout [11] and Beta distributed multiplicative noise.

3 Results

Figure 1 summarizes our results for the cart-pole domain. On the left hand side we compare PILCO with sparse spectrum GP regression [12] for the dynamics, with Deep-Pilco using BNN dynamics; one using binary dropout with dropout probability p=0.1, and the other using Log-Normal dropout with a Log-Uniform(0,1] prior. The BNN models are ReLU networks with two hidden layers of 200 units and a linear output layer. We do not discard older data during training. While fitting the models to experience data, we learn a homoscedastic measurement noise parameter

for each output dimension. This noise is used to corrupt the input to the policy. The policies are RBF networks with 30 units. The first interaction with the system corresponds to applying uniformly-at-random controls. On the right we see a comparison of various neural network policies in the same task, using the same BNN dynamics as on the left . The policy networks are ReLU networks with two hidden layers of 200 units. The results show that training complex neural network controllers provide marginally better performance than PILCO or Deep-PILCO with RBF controllers, without reducing data efficiency. The code used in these experiments is avalable at `https://github.com/juancamilog/kusanagi`.

Figure 2a illustrates the effect of the techniques on the more complicated double cart-pole swing-up task. The setup is similar to the cart-pole task, but we change the networks architecture slightly as the dynamics are more complex. The dynamics models are ReLU networks with 4 hidden layers of 200 units and a linear output layer. The policies are ReLU networks with four hidden layers of 50 units. In this case, the differences in performance are clearer. We are currently preparing experiments with these methods on more complex environments, including the AQUA robot [2].

While previous experiments combining PILCO with PEGASUS were unsuccessful [13], we found the combination of Deep-PILCO with this type of deterministic policy evaluation to be necessary for convergence when training neural network policies. We tried fixing the dropout masks across optimizer iterations, and using CRNs for the particle resampling step in Deep-PILCO. Both of these had a noticeable impact on convergence and reduced the computational cost of every policy update. We observed an improvement on the number of trials required for finding successful controller for the cart-pole swing-up task,and a reduction on the final accumulated cost.

References

[1] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, 2015.

[2] David Meger, Juan Camilo Gamboa Higuera, Anqi Xu, Philippe Giguere, and Gregory Dudek. Learning legged swimming gaits from experience. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2332–2338. IEEE, 2015.

[3] Yarin Gal, Rowan McAllister, and Carl E. Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, April 2016.

[4] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[5] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[6] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems 30*. 2017.

[7] Andrew Y Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

[8] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[9] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2575–2583. Curran Associates, Inc., 2015.

[10] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

[11] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. *arXiv preprint arXiv:1705.07832*, 2017.

[12] Joaquin QuiÃ±onero-Candela, Carl Edward Rasmussen, AnÃbal R Figueiras-Vidal, et al. Sparse spectrum gaussian process regression. *Journal of Machine Learning Research*, 11(Jun):1865–1881, 2010.

[13] Marc Peter Deisenroth. *Efficient reinforcement learning using Gaussian processes*, volume 9. KIT Scientific Publishing, 2010.