Synthesizing Neural Network Controllers with Probabilistic Model-Based Reinforcement Learning

Juan Camilo Gamboa Higuera, David Meger, and Gregory Dudek

***Abstract*— We present an algorithm for rapidly learning neural network policies for robotics systems. The algorithm follows the model-based reinforcement learning paradigm and improves upon existing algorithms: PILCO and a sample-based version of PILCO with neural network dynamics (Deep-PILCO). To improve convergence, we propose a model-based algorithm that uses fixed random numbers and clips gradients during optimization. We propose training a neural network dynamics model using variational dropout with truncated Log-Normal noise. These improvements enable data-efficient synthesis of complex neural network policies. We test our approach on a variety of benchmark tasks, demonstrating data-efficiency that is competitive with that of PILCO, while being able to optimize complex neural network controllers. Finally, we assess the performance of the algorithm for learning motor controllers for a six legged autonomous underwater vehicle. This demonstrates the potential of the algorithm for scaling up the dimensionality and dataset sizes, in more complex tasks.**

I. Introduction

Model-based reinforcement learning (RL) is an attractive framework for addressing the synthesis of controllers for robots of all kinds due to its promise of data-efficiency. An RL agent can use learned dynamics models to search for good controllers in simulation. This has the potential of minimizing costly trials on real robots. Minimizing interactions, however, means that datasets will often not be large enough to obtain accurate models. Bayesian models are very helpful in this situation. Instead of requiring an accurate model, the robot agent may keep track of a distribution over hypotheses of models that are compatible with its experience. Evaluating a controller then involves quantifying its performance over the model distribution. To improve its chances of working in the real world an effective controller should perform well, on average, on models drawn from this distribution. PILCO (Probabilistic Inference and Learning for COntrol) and Deep-PILCO are successful applications of this idea.

PILCO [1] uses Gaussian Process (GP) models to fit one-step dynamics and networks of radial basis functions (RBFs) as feedback policies. PILCO has been shown to perform very well with little data in 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 $O(Dn^3)$ and long-term predictions scale $O(D^3n^2)$, where n is the dataset size and D is the number of state dimensions, limiting its applicability only to scenarios with small datasets and low dimensionality.

The authors are part of the Center for Intelligent Machines and the School of Computer Science, McGill University, Montreal, Canada. This work was supported by NSERC through funding for the NSERC Canadian Field Robotics Network {gamboa,dmeger,dudek}@cim.mcgill.ca

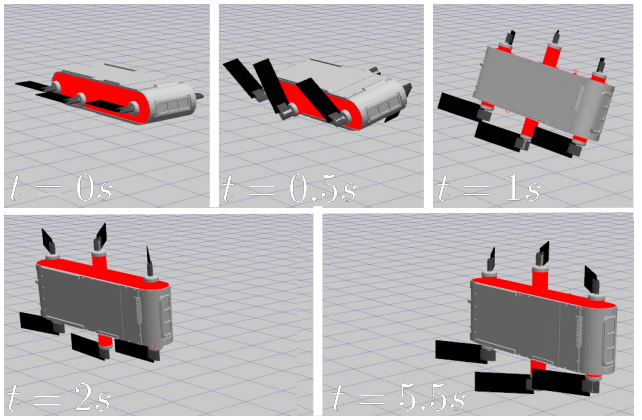


Fig. 1: The AQUA robot executing a *6-leg knife edge* maneuver. The robot starts in its resting position and must swim forward at a constant depth while stabilizing a roll angle of 90 degrees. The sequence of images illustrates a controller obtained with our proposed approach.

Deep-PILCO [3] aims to address these limitations by employing Bayesian Neural Networks (BNNs), implemented via binary dropout [4], [5]. Deep-PILCO performs a sampling-based procedure for simulating with BNN models of the dynamics. Policy search and model learning are done via stochastic gradient optimization, which scales more favorably to larger datasets and higher dimensionality. Deep-PILCO has been shown to result in better policies for a cart-pole swing-up benchmark task, but show reduced data efficiency when compared with PILCO. We extend on the results of [3] by:

- Modifying the simulation procedure to incorporate the use of fixed random numbers for policy optimization
- Clipping gradients to stabilize optimization with back-propagation through time (BPTT)
- Using BNNs with multiplicative parameter noise where the noise distribution is adapted from data [6]

We show how these improvements allow us to optimize neural network controllers with Deep-PILCO, while matching the data efficiency of PILCO on the cart-pole swing-up task; i.e. learning a successful controller with the same amount of experience. We also show how training stochastic policies (implemented as BNNs) can be beneficial for the convergence of robust policies. Finally, we demonstrate how these methods can be applied for learning swimming controllers for a 6 legged autonomous underwater vehicle.

II. Related Work

Dynamics models have long been a core element in the modeling and control of robotic systems. Trajectory optimization approaches [7], [8], [9] can produce highly effective controllers for complex robotic systems when precise analytical models are available. For complex and stochastic systems such as swimming robots, classical models are less reliable. In these cases, either performing online system identification [10] or learning complete dynamics models from data has proven to be effective, and can be integrated tightly with model-based control schemes [11], [12], [13], [14].

Multiple works have applied Deep RL methods to learn various continuous control tasks [15], [16], including full-body control of humanoid characters [17]. These methods do not assume a known reward function, estimating the value of each action from experience. Along with their model-free nature, this results in lower data efficiency compared with the methods we consider here, but there are ongoing efforts to connect model-based and model-free approaches [18].

The most similar works to our own are those which use probabilistic dynamics models for policy optimization. Locally linear controllers can be learned in this fashion, for example by extending the classical Differential Dynamic Programming (DDP) [19] method or Iterative LQG [20] to use GP models. For more complex robots, it is desirable to learn complex non-linear policies using the predictions of learned dynamics. Black-DROPS [21] has recently shown promising performance competitive with the gradient-based PILCO [1] for training GP and NN policies using GP dynamics models. As yet, we are only aware of BNNs being used in the policy learning loop within Deep-PILCO [3], which is the method we directly improve upon. Our approach is the first model-based RL approach to utilize BNNs for both the dynamics as well as the policy network.

III. Problem Statement

We focus on model-based policy search methods for episodic tasks. We consider systems that can be modeled with discrete-time dynamics $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$, where f is unknown, with states $\mathbf{x}_t \in \mathbb{R}^D$ and controls $\mathbf{u}_t \in \mathbb{R}^U$, indexed by time-step t. The goal is to find the parameters θ of a policy π_θ that minimize a task-dependent cost function c accumulated over a finite time horizon H,

$$\arg\min_{\theta} \quad J(\theta) = \mathbb{E}_\tau \left\{ \sum_{t=1}^{H} c(\mathbf{x}_t) \;\middle|\; \theta \right\}. \tag{1}$$

The expectation in our case is due to not knowing the true dynamics f, which induces a distribution over trajectories $p(\tau) = p(\mathbf{x}_1, \mathbf{u}_1, ..., \mathbf{x}_H, \mathbf{u}_H | f)$. The objective could be minimized by black-box optimization or likelihood ratio methods, obtaining trajectory samples directly from the target system. However, such methods are known to require a large number of evaluations, which may be impractical for applications with real robot systems. An alternative is to use experience to fit a model of the dynamics $f_{\mathbf{w}}$ and use it to estimate the objective in Eq. (1). Alg. (1) describes a sketch for model-based optimization methods. A goal of these methods is *data-efficiency*: to use as little real-world experience as possible. Since we consider fixed horizon tasks, data-efficiency can be measured in the number of episodes, or trials, until the task is successfully learned.

Algorithm 1 Episodic Model-Based RL

1: Initialize parameters θ, $\mathbf{w}$ and dataset $\mathcal{D}$
2: **for** episode e in $1...N_{\text{trials}}$ **do**
3: Obtain τ_e by executing π_θ for H steps on robot
4: Append τ_e to $\mathcal{D}$
5: Use $\mathcal{D}$ to update $\mathbf{w}$ ▷ model learning
6: Use $f_{\mathbf{w}}$ to minimize $J(\theta)$, update θ ▷ policy optimization
7: Return $\pi_\theta, f_{\mathbf{w}}$

IV. Background

A. Learning a dynamics model with BNNs

A key to data-efficiency is avoiding *model bias* [22], [23], i.e. optimizing Eq. (1) with a model that makes bad predictions with high confidence. BNNs address model bias by using the posterior distribution over their parameters. Given a model $f_{\mathbf{w}}$ with parameters $\mathbf{w}$ and a dataset $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$ we'd like to use the posterior $p(\mathbf{w}|\mathcal{D})$ to make predictions at new test points. This distribution represents the uncertainty about the true value of $\mathbf{w}$, which induces uncertainty on the model predictions: $p(\mathbf{y}) = \int p(\mathbf{y}|f_{\mathbf{w}}, \mathbf{x}) p(\mathbf{w}|\mathcal{D}) d\mathbf{w}$, where $\mathbf{y}$ is the prediction at test point $\mathbf{x}$. Using the true posterior for predictions on a neural network is intractable. Fortunately, various methods based on variational inference exist, which use tractable approximate posteriors and Monte Carlo integration for predictions [5], [24], [25], [26], [27]. Fitting is done by minimizing the Kullback-Leibler (KL) divergence between the true and the approximate posterior, which can done by optimizing the objective

$$\mathcal{L}(\mathbf{w}) = -\mathcal{L}_{\mathcal{D}}(\mathbf{w}) + D_{KL}\left(q(\mathbf{w}) | p(\mathbf{w})\right) \tag{2}$$

where $\mathcal{L}_{\mathcal{D}}$ is the expected value of the likelihood $p\left(\mathcal{D}|\mathbf{w}\right)$, $q(\mathbf{w})$ is the approximate posterior and $p(\mathbf{w})$ is a user-defined prior on the parameters. These methods usually set $q(\mathbf{w})$ as a deterministic transformation of noise samples $\mathbf{w} = g(\psi, \mathbf{z}_{\mathbf{w}})$, $\mathbf{z}_{\mathbf{w}} \sim p(\mathbf{z}_{\mathbf{w}})$, where ψ are the parameters of the posterior [25]. For example, in binary dropout g multiplies the weights matrices for each layer of the network with the dropout masks $\mathbf{z}_{\mathbf{w}}$, consisting of diagonal noise matrices with entries drawn from a Bernoulli distribution with dropout probability p [4]. To fit the dynamics model, we build the dataset $\mathcal{D}$ of tuples $\langle (\mathbf{x}_t, \mathbf{u}_t), \Delta_t \rangle$; where $(\mathbf{x}_t, \mathbf{u}_t) \in \mathbb{R}^{D+U}$ are the state-action pairs that we use as input to the dynamics model, and $\Delta_t = \mathbf{x}_t - \mathbf{x}_{t-1} \in \mathbb{R}^X$ are the changes in state after applying action $\mathbf{u}_t$. We fit the model by minimizing the objective in Eq.(2) via stochastic gradient descent.

B. Policy optimization with learned models

To estimate the objective function in Eq. (1) we base our approach on Deep-PILCO [3], which we summarize in Alg. (2). For every optimization iteration, the algorithm draws particles consisting of an initial state and a set of

Algorithm 2 Policy search with Deep-PILCO

1: **for** j in $1...N_{\text{opt}}$ **do**
2: Sample K particles $\{(\mathbf{x}_1^{(k)}, f_{\mathbf{w}^{(\mathbf{k})}})|1 \leq k \leq K\}$
3: **for** t in $1...H$ **do**
4: **for** k in $1...K$ **do**
5: Evaluate policy $\mathbf{u}_t^{(k)} = \pi_\theta\left(\mathbf{x}_t^{(k)}\right)$
6: Propagate state $\mathbf{x}_{t+1}^{(k)} = f_{\mathbf{w}^{(\mathbf{k})}}\left(\mathbf{x}_t^{(k)}, \mathbf{u}_t^{(k)}\right)$
7: Fit mean $\mu_{\mathbf{x}_{t+1}}$ and covariance $\Sigma_{\mathbf{x}_{t+1}}$
8: Resample $\mathbf{x}_{t+1}^{(k)}$ from $\mathcal{N}(\mu_{\mathbf{x}_{t+1}}, \Sigma_{\mathbf{x}_{t+1}})$
9: Evaluate objective $J(\theta) = \frac{1}{K}\sum_{k=1}^{K}\sum_{t=1}^{H} c(\mathbf{x}_t^{(k)})$
10: Compute gradient estimate $\nabla_\theta J(\theta)$
11: Update θ by stochastic gradient descent step

weights sampled from $q(\mathbf{w})$, as shown in line 2. For the models used in [3] and this work, sampling weights is equivalent to sampling dropout masks $\mathbf{z}_\mathbf{w}$. The loop in lines 4 to 6 can be executed in parallel using batch processing. This algorithm requires the task cost function c to be known and differentiable. Deep-PILCO uses back-propagation through time (BPTT) to estimate the policy gradients $\nabla_\theta J(\theta)$.

V. Improvements to Deep-PILCO

Here we describe the changes we have done to Deep-PILCO that were crucial for improving its data-efficiency and obtaining the results we describe in Sec. VI. Our changes are summarized in Alg. (3). This algorithm can still be executed efficiently using batch processing with state-of-the art deep learning frameworks.

A. Common random numbers for policy evaluation

The convergence of Algorithms (2) and (3) is highly dependent on the variance of the estimated gradient $\nabla_\theta J(\theta))$. In this case, the variance of the gradients is dependent on the sources of randomness for simulating trajectories: the initial state samples $\mathbf{x}_1$, the multiplicative noise masks $\mathbf{z}_\mathbf{w}^{(k)}, \mathbf{z}_\theta^{(k)}$, and the random numbers used for re-sampling $\mathbf{z}_t^{(k)}$. A common variance reduction technique used in stochastic optimization is to fix random numbers during optimization [28]. Using common random numbers (CRNs) reduces variance in two ways: gradient evaluations become deterministic and evaluations over different values for the optimization variable become correlated. We introduce CRNs by drawing all the random numbers we need for simulating trajectories at the beginning of the policy optimization (lines 1 to 3 in Alg. (3)) and keeping them fixed as the policy parameters are updated. This is possible because we use BNNs that rely on the re-parametrization trick [25] for evaluation. This is effective in reducing variance and improving convergence, but it may introduce bias. A simple way to deal with bias is to increase the number of particles K used for gradient evaluation. We increased K from 10, the number used in [3], to 100 for our experiments, and found it to improve convergence with small penalty on running time.

Fixing random numbers in the context of policy search is known as the PEGASUS[1] algorithm [29]. PEGASUS

[1]Policy Evaluation-of-Goodness And Search Using Scenarios

Algorithm 3 Our method: Deep-PILCO with PEGASUS evaluation and gradient clipping

1: Sample noise for dynamics $\{\mathbf{z}_\mathbf{w}^{(k)}|1 \leq k \leq K\}$
2: Sample noise for policy $\{\mathbf{z}_\theta^{(k)}|1 \leq k \leq K\}$
3: Sample state noise $\{\mathbf{z}_t^{(k)}|1 \leq k \leq K, 1 \leq t \leq H\}$
4: **for** j in $1...N_{\text{opt}}$ **do**
5: Sample K particles $\{\mathbf{x}_1^{(k)}|1 \leq k \leq K\}$
6: **for** t in $1...H$ **do**
7: **for** k in $1...K$ **do**
8: $\theta^{(k)} = g_1(\theta, \mathbf{z}_\theta^{(k)})$
9: $\mathbf{w}^{(k)} = g_2(\mathbf{w}, \mathbf{z}_\mathbf{w}^{(k)})$
10: Evaluate policy $\mathbf{u}_t^{(k)} = \pi_{\theta^{(k)}}\left(\mathbf{x}_t^{(k)}\right)$
11: Propagate state $\mathbf{x}_{t+1}^{(k)} = f_{\mathbf{w}^{(\mathbf{k})}}\left(\mathbf{x}_t^{(k)}, \mathbf{u}_t^{(k)}\right)$
12: Fit mean $\mu_{\mathbf{x}_{t+1}}$ and covariance $\Sigma_{\mathbf{x}_{t+1}}$
13: **for** k in $1...K$ **do**
14: $\mathbf{x}_{t+1}^{(k)} = \mu_{\mathbf{x}_{t+1}} + \Sigma_{\mathbf{x}_{t+1}}\mathbf{z}_t^{(k)}$
15: Evaluate objective $J(\theta) = \frac{1}{K}\sum_{k=1}^{K}\sum_{t=1}^{H} c(\mathbf{x}_t^{(k)})$
16: Compute gradient estimate $\nabla_\theta J(\theta)$
17: **if** $\nabla_\theta J(\theta) > \nu$ **then**
18: $\nabla_\theta J(\theta) \leftarrow \nu \frac{\nabla_\theta J(\theta)}{||\nabla_\theta J(\theta)||}$
19: Update θ by stochastic gradient descent step

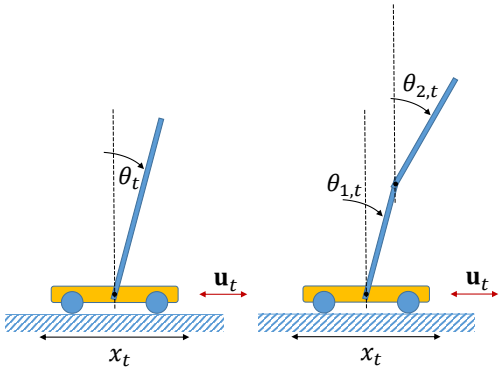
consists of transforming a given Markov Decision Process (MDP) into "an equivalent one where all transitions are deterministic" by assuming access to a *deterministic simulative model* of the MDP. A deterministic simulative model is one that has no internal random number generator, so any random numbers that are needed must be given to it as input. This is the case when using BNNs models. PEGASUS provides theoretical justification to our approach, particularly in that to decrease the upper bound on the error of estimates of $J(\theta)$ using CRNs it suffices to increase K.

B. Stabilization for back-propagation through time

As noted in [3], the recurrent application of BNNs in Algortihm 2 can be interpreted as a Recurrent Neural Network (RNN) model. As such, Deep-PILCO is prone to suffer from vanishing and exploding gradients when computing them via BPTT [30], 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 opted to deal with these problems by using ReLU activations for the policy and dynamics model, and clipping the gradients to have a norm of at most ν. We show the effect of various settings of the clipping value ν on the convergence of policy search in Fig. (3b) .

C. BNN models with Log-Normal multiplicative noise

We focused on methods that use multiplicative noise on the activations (e.g. binary dropout) because of their simplicity and computational efficiency. Deep-PILCO with binary dropout requires tuning the dropout probability to a value appropriate for the model size. We experimented with various BNN models [6], [25], [26], [27] to enable learning the dropout probabilities from data. The best performing



(a) Cart-pole task (b) Double cart-pole task

Fig. 2: Benchmark tasks used in Sec. VI. In both tasks, the tip of the pendulum starts downright, with the cart centered at $x = 0$. The goal is to balance the tip of the pole at its highest possible location, while keeping the cart at $x = 0$. This occurs when $\theta = 0$ for the cart-pole, and when $\theta_1 = 0, \theta_2 = 0$ in the double cart-pole.

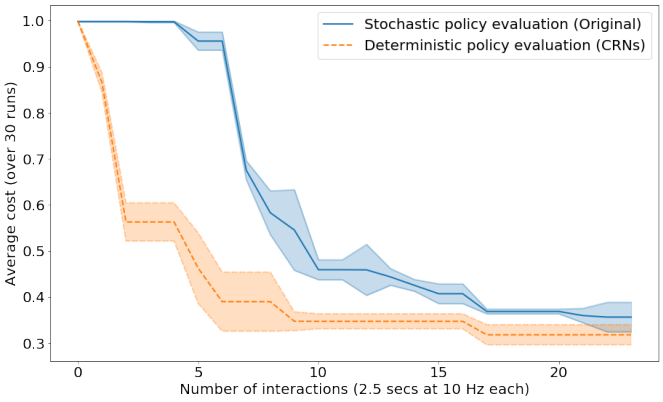
method in our experiments was using truncated Log-Normal dropout with a truncated log-uniform prior $\text{LogU}_{[-10,0]}$. This choice prior causes the multiplicative noise $\mathbf{z_w}$ to be constrained to values between 0 and 1 [6].

D. Training neural network controllers

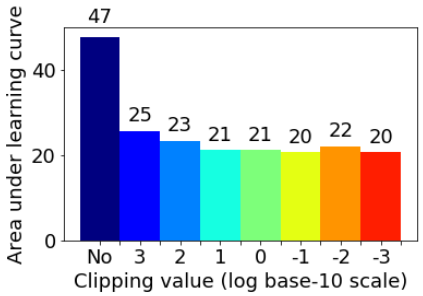
While Deep-PILCO had been limited to training single-layer Radial Basis Function policies, the application of gradient clipping and CRNs allows stable training of deep neural network policies, opening the door for richer behaviors. We found that adding dropout to the policy networks improves performance. During policy evaluation, we sample policies the same way as we do for dynamics models: a policy sample corresponds to a set of dropout masks $\mathbf{z}_\theta^{(k)}$. Thus each simulated state particle has a corresponding dynamics model and policy, which remain fixed during policy optimization. This can be interpreted as attempting to learn a distribution of controllers that are likely to perform well over plausible dynamics models. We make the policy stochastic during execution on the target system by re-sampling the policy dropout mask $\mathbf{z}_\theta$ at every time step. This provides some amount of exploration that we found beneficial.

VI. Results

We tested the improvements, described in Section V, on two benchmark scenarios: swinging up and stabilizing an inverted pendulum on a cart, and swinging up and stabilizing a double pendulum on a cart (see Fig. (2)). The first task was meant to compare performance on the same experiment as [3]. We chose the second scenario to compare the methods with a harder long-term prediction task; due to the chaotic dynamics of the double-pendulum. In both cases, the system is controlled by applying a horizontal force $\mathbf{u}$ to the cart. We also evaluate our approach on the gait learning tasks for an underwater hexapod robot [2] to demonstrate the applicability of our approach for locomotion tasks on complex robot systems. We use the ADAM optimizer [31] for model fitting and policy optimization, with the default parameters



(a) Comparison of the effect of introducing CRNs



(b) Effect of clipping gradients

Fig. 3: (a) Illustrates the benefit of fixing random numbers for policy evaluation (Alg. (3)) vs. stochastic policy evaluations (Alg. (2)). In (b) we show the area under the learning curve for the cart-pole task for various gradient clipping values (lower is better).

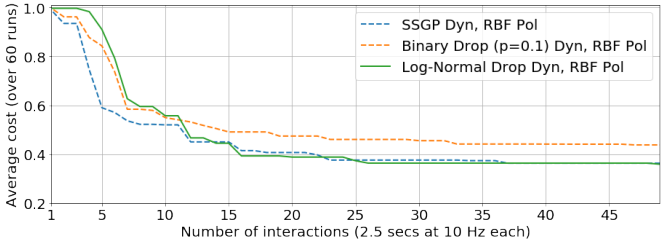
suggested by the authors, and report the best results obtained after manual hyper-parameter tuning.

A. Cart-pole swing-up task

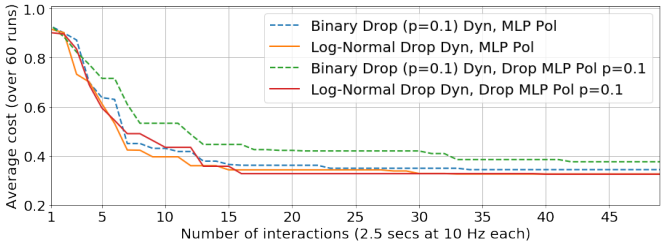
While previous experiments combining PILCO with PEGASUS were unsuccessful [23], we found its application to Deep-PILCO to result in a significant improvement on convergence when training neural network policies. Fig. (3a) shows how the use of CRNs (Deterministic policy evaluation) results in faster convergence than the original Deep-PILCO formulation (Stochastic policy evaluation), which only matches the cost of our approach after around the 20th trial. These experiments were done with a learning rate of 10^{-4} and clipping value $\nu = 1.0$. Fig. (3b) illustrates the effect of gradient clipping for different values of ν for Alg. (3). The area under the learning curve gives us an idea of the speed of convergence as the clipping value changes. The trend is that any value of gradient clipping made a large improvement over not clipping at all and that the specific choice of clipping values was highly stable.

Fig. (4) summarizes our results for the cart-pole domain[2]. Fig. (4a) illustrates the difference in performace between PILCO using sparse spectrum GP (SSGP) regression [32] for the dynamics and two versions of Deep-PILCO using BNN dynamics: one using binary dropout with dropout probability

[2]The code used in these experiments is available at `https://github.com/juancamilog/kusanagi`.



(a) Cart-pole RBF Policies



(b) Cart-pole Deep Policies

Fig. 4: Cost per trial on the cart-pole swing-up task. In (a), we compare different dynamics models for learning RBF policies. (b) compares BNN models for learning NN policies, showing how our approach matches the data-efficiency of PILCO with better final performance.

$p = 0.1$, and the other using Log-Normal dropout with a truncated log-uniform $\text{LogU}_{[-10,0]}$. The BNN models are ReLU networks with two hidden layers of 200 units and a linear output layer. The models predict heteroscedastic noise, which is used to corrupt the input to the policy during simulation. We used data from all previous episodes for model learning after each trial. The initial experience was gathered with a single execution of a policy that selects actions uniformly-at-random. The learning rate was set to 10^{-4} for model learning and 10^{-3} for policy optimization. The policies were RBF networks with 30 units. Fig. (4b) provides a comparison of different BNN dynamics models when training neural network policies. The policy networks are ReLU networks with two hidden layers of 200 units. For BNN policies (Drop MLP) we set a constant dropout probability $p = 0.1$. Note that our method is able to train neural network controllers with better performance (lower cost) than either PILCO or Deep-PILCO with RBF controllers, within a similar number of trials. Using truncated Log-Normal dropout (Log-Normal Drop Dyn) for learning a stochastic policy (Drop MLP) results in the best performance for the cart-pole task.

B. Double pendulum on cart swing-up task

Fig. (5) illustrates the effect of learning neural network controllers on the more complicated double cart-pole swing-up task. We were unable to get Alg. (2) with RBF policies to converge in this task. The setup is similar to the cart-pole task, but we change the network architectures 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. The learning rate for policy learning was set to 10^{-4}. The initial experience was comes

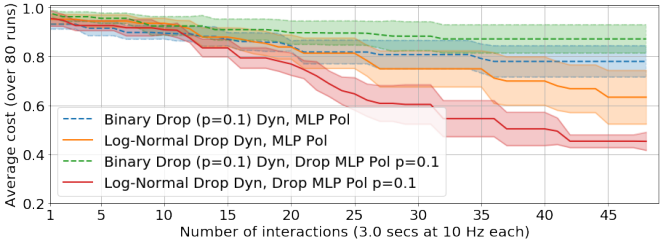


Fig. 5: Cost per trial on the double cart-pole swing-up task. The shaded regions correspond to half a standard deviation. This demonstrates the benefit of using Log-Normal multiplicative noise for the dynamics with dropout regularization for the policies

from 2 runs with random actions. Here the differences in performance are more pronounced: our method converges after 42 trials, corresponding to 126 s of experience at 10 Hz. This is close to the 84 s at 13.3 Hz reported in [23]. We see that the combination of BNN dynamics (Log-Normal Drop Dyn) and a BNN policy (Drop MLP Pol) results in the least number of trials for achieving the lowest cost.

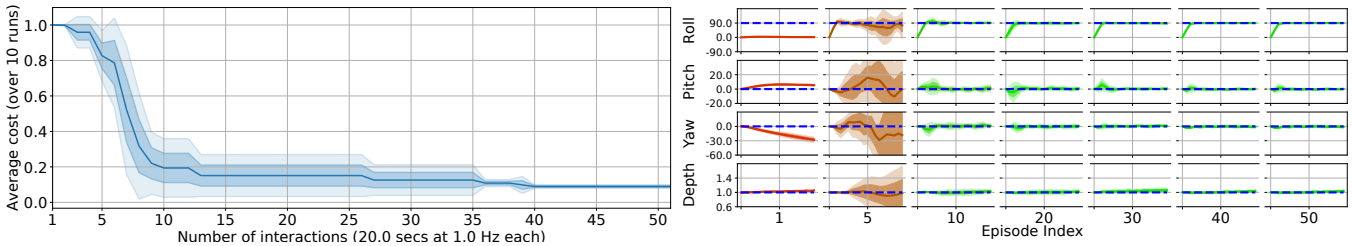
C. Learning swimming gaits on an underwater robot.

These tasks consist of finding feedback controllers for controlling the robot's 3D pose via periodic motion of its legs. Fig. (1) illustrates the execution of a gait learned using our methods. The robot's state space consists of readings from its inertial measurement unit (IMU), its depth sensor and motor encoders. To compare with previously published results, the action space is defined as the parameters of the periodic leg command (PLC) pattern generator [2], with the same constraints as prior work. We conducted experiments on the following control tasks[3]:

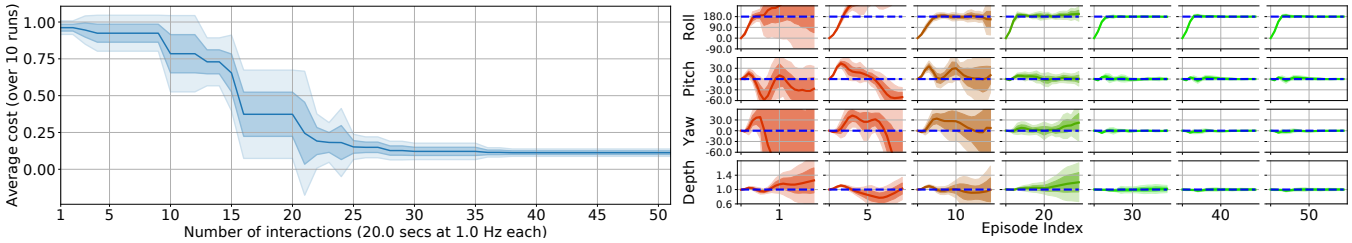
1) *knife edge*: Swimming straight-ahead with $90\,\text{deg}$ roll
2) *belly up*: Swimming straight-ahead with $180\,\text{deg}$ roll
3) *corkscrew*: Swimming straight-ahead with $120\,\text{deg}$ rolling velocity (anti-clockwise)
4) *1 m depth change*: Diving and stabilizing 1 meter below current depth.

There were two versions of these experiments. In the first one, which we call *2-leg tasks*, the robot controls only the amplitudes and offsets of the two back legs (4 control dimensions). Its state corresponds to the angles and angular velocities, as measured by the IMU, and the depth sensor measurement (7 state dimensions). In the second version, the robot controls amplitudes and offsets and phases for all 6 legs (18 control dimensions). In this case, the state consists of the IMU and depth sensor readings plus the leg angles as measured form the motor encoders (13 state dimensions). We transform angle dimensions into their complex representation before passing the state as input to the dynamics model and policy, as described in [23]. We trained dynamics models and policies with 4 hidden layers of 200 units each. The dynamics models use truncated Log-Normal dropout and we enable dropout for the policy with $p = 0.1$. We used a learning

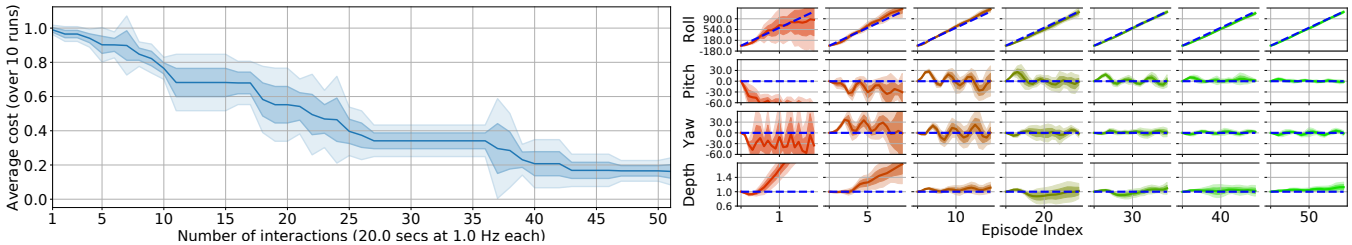
[3]The code used for these experiments and video examples of other learned gaits are available at `https://github.com/mcgillmrl/robot_learning`.



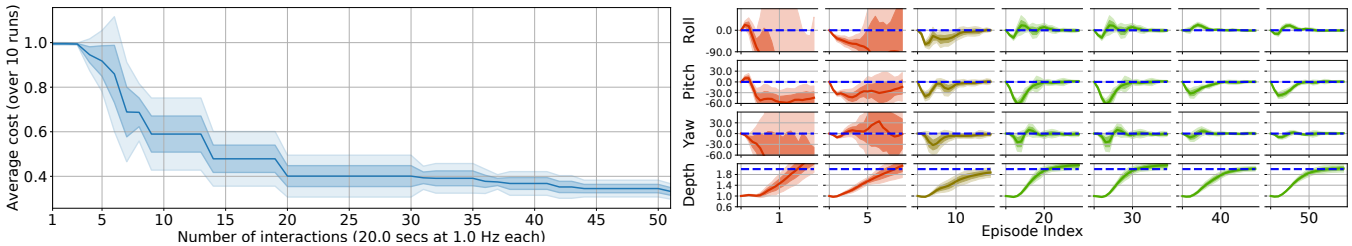
(a) 2-Leg Knife edge



(b) 2-Leg Belly up



(c) 2-Leg Corkscrew



(d) 2-Leg Depth change

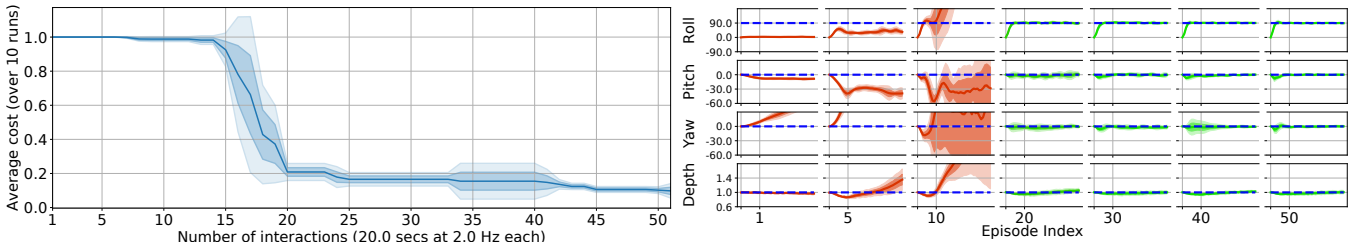
Fig. 6: Learning curve and the evolution of the trajectory distribution as learning progresses for 2-leg tasks. The robot learns to control its pose by setting the appropriate amplitudes and leg offset angles for its back 2 legs. The dashed lines represent the desired target states. Additional results and videos of these behaviours available at `https://github.com/mcgillmrl/robot_learning`

rate of 10^{-4} and clip gradients to $\nu = 1.0$. The experience dataset is initialized with 5 random trials, common to all the tasks with the same state and action spaces.

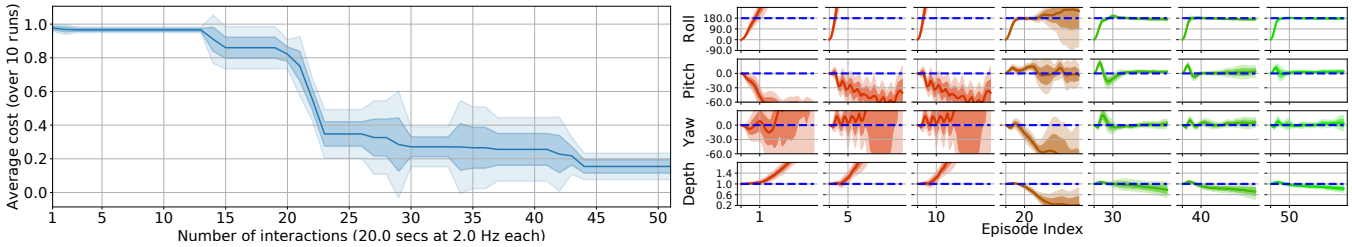
Fig. (6) and (7) show the results of gait learning in the simulation environment described in [2]. In addition to learning curves on the left of each task panel, we show detailed state telemetry for selected learning episodes on the right to provide intuition on stability and learning progression. The shaded regions represent the variance of the trajectory distributions on the target system over 10 different runs. In each case attempted, our method was able to learn effective swimming behavior, to coordinate the motions of multiple flippers and overcome simulated hydrodynamic effects without any prior model. For the 2-leg tasks, our method obtains successful policies in 10-20 trials, a number competitive with results reported in [2]. We obtained successful controllers for the depth-change task, which was unsuccessful in prior work. The 6-leg tasks, with their considerably higher-dimensional state and action spaces, take roughly double the number of trials. But all tasks still converged by trial 50, which remains practical for real deployment.

VII. Conclusion

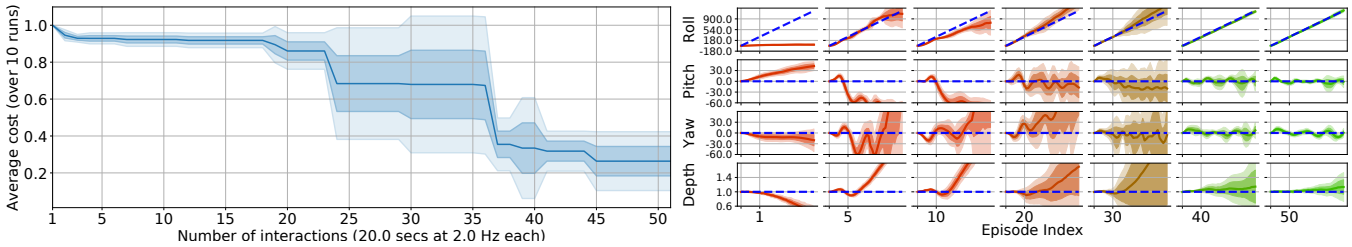
We have presented improvements to a probabilistic model-based reinforcement learning algorithm, Deep-PILCO, to enable fast synthesis of controllers for robotics applications. Our algorithm is based on treating neural network models trained with dropout as an approximation to the posterior distribution of dynamics models given the experience data. Sampling dynamics models from this distribution helps in avoiding model-bias during policy optimization; policies are optimized for a finite sample of dynamics models, obtained



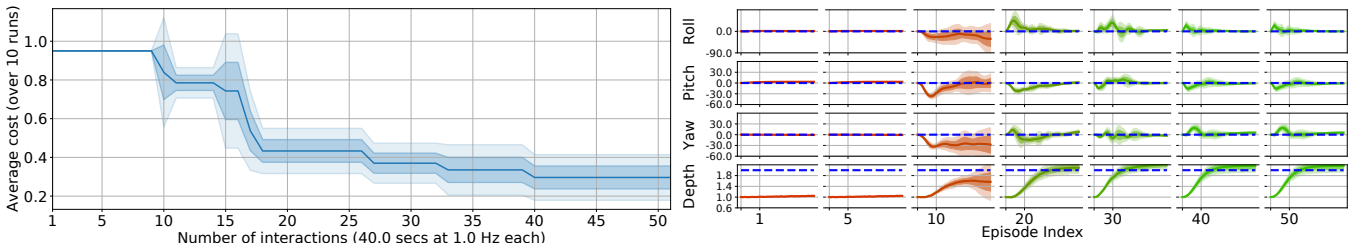
(a) 6-Leg Knife edge



(b) 6-Leg Belly up



(c) 6-Leg Corkscrew



(d) 6-Leg Depth change

Fig. 7: Learning curve and the evolution of the trajectory distribution as learning progresses for 6-leg tasks. In this case, the robot is trying to control the amplitudes, leg angle offsets, and phase offsets for all 6 legs. The algorithm takes longer to converge in this case, when compared to the 2-leg tasks. This is possibly due to the larger state and action spaces (13 state dimensions + 18 action dimensions). Nevertheless, this demonstrates that the algorithm can scale to higher dimensional problems.

through the application of dropout noise masks. Our changes enable training of neural network controllers, which we demonstrate to outperform RBF controllers on the cart-pole swing-up task. We obtain competitive performance on the task of swing-up and stabilization of a double pendulum on a cart. Finally, we demonstrated the usefulness of the algorithm on the higher dimensional tasks of learning gaits for pose stabilization for a six legged underwater robot. We replicate previous results [2] where we control the robot with 2 flippers, and provide new results on learning to control the robot using all 6 legs, now including phase offsets.

REFERENCES

[1] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.

[2] D. Meger, J. C. G. Higuera, A. Xu, P. Giguere, and G. Dudek, "Learning legged swimming gaits from experience," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015.

[3] Y. Gal, R. McAllister, and C. E. Rasmussen, "Improving PILCO with Bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop, ICML*, 2016.

[4] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, 2014.

[5] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

[6] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, "Structured bayesian pruning via log-normal multiplicative noise," in *Advances in Neural Information Processing Systems*, 2017.

[7] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*. Elsevier, 1970.

[8] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *Proceedings of the 1st International Conference on Informatics in Control, Automation and*

Robotics, 2004.

[9] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2014.

[10] A. D. Marchese, R. Tedrake, and D. Rus, "Dynamics and trajectory optimization for a soft spatial fluidic elastomer manipulator," *International Journal of Robotics Research*, 2015.

[11] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, Nov 2011.

[12] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," *Lazy learning*, pp. 75 – 113, 1997.

[13] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Proceedings of Neural Information Processing Systems (NIPS)*, 2006.

[14] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems*, 2014, pp. 1071–1079.

[15] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016.

[16] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *Advances in Neural Information Processing Systems*, 2015.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[18] S. Bansal, R. Calandra, S. Levine, and C. Tomlin, "MBMF: model-based priors for model-free reinforcement learning," *CoRR*, vol. abs/1709.03153, 2017.

[19] Y. Pan and E. Theodorou, "Probabilistic differential dynamic programming," in *Advances in Neural Information Processing Systems*, 2014.

[20] G. Lee, S. S. Srinivasa, and M. T. Mason, "GP-ILQG: data-driven robust optimal control for uncertain nonlinear dynamical systems," *CoRR*, vol. abs/1705.05344, 2017.

[21] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-box data-efcient policy search for robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[22] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *Proceedings of International Conference on Robotics and Automation*, vol. 4, 1997.

[23] M. P. Deisenroth, *Efficient reinforcement learning using Gaussian processes*. KIT Scientific Publishing, 2010.

[24] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *International Conference on Machine Learning*, 2015.

[25] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in Neural Information Processing Systems*, 2015.

[26] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *International Conference on Machine Learning*, 2017.

[27] Y. Gal, J. Hron, and A. Kendall, "Concrete dropout," in *Advances in Neural Information Processing Systems*, 2017.

[28] N. L. Kleinman, J. C. Spall, and D. Q. Naiman, "Simulation-based optimization with stochastic approximation using common random numbers," *Management Science*, 1999.

[29] A. Y. Ng and M. Jordan, "Pegasus: A policy search method for large mdps and pomdps," in *Proceedings of the Sixteenth conference on Uncertainty in Artificial Intelligence*, 2000.

[30] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, 2013.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[32] M. Lázaro-Gredilla, J. Quiñonero Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal, "Sparse spectrum gaussian process regression," *Journal of Machine Learning Research*, 2010.