# Improving PILCO with Bayesian Neural Network Dynamics Models

Yarin Gal and Rowan Thomas McAllister and Carl Edward Rasmussen[1]

*Abstract*— **Model-based reinforcement learning (RL) allows an agent to discover good policies with a small number of trials by generalising observed transitions. Data efficiency can be further improved with a *probabilistic* model of the agent's ignorance about the world, allowing it to choose actions under uncertainty. Bayesian modelling offers tools for this task, with PILCO [1] being a prominent example, achieving state-of-the-art data efficiency on low dimensional RL benchmarks. But PILCO relies on Gaussian processes (GPs), which prohibits its applicability to problems that require a larger number of trials to be solved. Further, PILCO does not consider temporal correlation in model uncertainty between successive state transitions, which results in PILCO underestimating state uncertainty at future time steps [2]. In this paper we extend PILCO's framework to use Bayesian deep dynamics models with approximate variational inference, allowing PILCO to scale linearly with number of trials and observation space dimensionality. Using particle methods we sample dynamics function realisations, and obtain lower cumulative cost than PILCO. We give insights into the modelling assumptions made in PILCO, and show that moment matching is a crucial simplifying assumption made by the model. Our implementation can leverage GPU architectures, offering faster running time than PILCO, and will allow structured observation spaces to be modelled (images or higher dimensional inputs) in the future.**

## I. INTRODUCTION

Reinforcement learning (RL) algorithms learn control tasks via trial and error, much like a child learning to ride a bicycle [3]. But trials of real world control tasks often involve time and resources we wish not to waste. Alternatively, the number of trials might be limited due to wear and tear of the system, making data-efficiency critical. Exploration techniques such as Thompson sampling [4] can help learn better policies faster. But a much more drastic improvement in data efficiency can be achieved by modelling the system dynamics [5]. A dynamics model allows the agent to generalise its knowledge about the system dynamics to other, unobserved, states. *Probabilistic* dynamics models allow an agent to consider transition uncertainty throughout planning and prediction, improving data efficiency even further. PILCO [1], for example, is a data-efficient probabilistic model-based policy search algorithm. PILCO analytically propagates uncertain state distributions through a Gaussian process (GP) dynamics model. This is done by recursively feeding the output state distribution (*output uncertainty*) of one time step as the input state distribution (*input uncertainty*) of the next time step, until a fixed time horizon $T$. This allows the agent to consider the long-term consequences (expected cumulative cost) of a particular controller parametrisation w.r.t. all plausible dynamics models. PILCO relies on GPs, which work extremely well with small amounts of low dimensional data, but scale cubically with the number of trials. Further, PILCO's distribution propagation adds a squared term in the observation space dimensionality, making it hard to scale the framework to high dimensional observation spaces. This makes it difficult to use PILCO with tasks that require a larger number of trials. Further, PILCO does not consider temporal correlation in model uncertainty between successive state transitions. This means that PILCO underestimates state uncertainty at future time steps [2], which can lead to diminished performance.

Here we attempt to answer these shortcomings by replacing PILCO's Gaussian process with a Bayesian deep dynamics model, while maintaining the framework's probabilistic nature and its data-efficiency benefits. But this task poses several interesting difficulties. First, we have to handle *small data*, and neural networks are notoriously known for their tendency to overfit. Furthermore, we must retain PILCO's ability to capture 1) dynamics model output uncertainty and 2) input uncertainty. Output uncertainty can be captured with a Bayesian neural network (BNN), but end-to-end inference poses a challenge. Input uncertainty in PILCO is obtained by analytically propagating a state distribution through the dynamics model. But this can neither be done analytically with NNs nor with BNNs. Our solution to handling output uncertainty relies on dropout as a Bayesian approximation to the dynamics model posterior [6]. This allows us to use techniques proven to work well in the field, while following their probabilistic Bayesian interpretation. Input uncertainty in the dynamics model is captured using particle techniques. To do this we solve the difficulties encountered by [7] while attempting this particle technique with PILCO in the past. Interestingly, unlike PILCO our approach allows us to sample dynamics functions, required for accurate variance estimates of future state distributions.

Our approach has several benefits compared to the existing PILCO framework. First, as we require lower time complexity (linear in trials and observation space dimensionality), we can scale our algorithm well to tasks that necessitate more trials for learning. We demonstrate this by analysing the running time of our algorithm compared to PILCO's on a standard benchmark. Second, unlike PILCO we can sample dynamics function realisations, resulting in better cumulative cost than PILCO's on the cartpole swing-up task (a 25% reduction in cumulative cost). The use of a NN dynamics model comes at a price though, where we need to use a slightly higher number of trials than PILCO. Our approach offers insights into the modelling assumption made in PILCO, such as the consequences of performing

[1] Department of Engineering, University of Cambridge, [yg279, rtm26, cer54]@cam.ac.uk

moment matching. Lastly, our model can be seen as a Bayesian approach to performing data efficient deep RL. In the experiments section we compare our approach to that of recent deep RL algorithms [8], [9], showing orders of magnitude improvement in data efficiency on these.

The following sections are structured as follows. First we outline the original PILCO algorithm (Section II) before introducing our algorithm – an adaptation of PILCO (Section III). We then describe our experiments using the cartpole swing-up task (Section IV) used to compare data-efficiency against PILCO, and discuss our insights (Section V). Finally, we discuss future work (Section VI).

## II. PILCO

In this section we outline the PILCO (*probabilistic inference for learning control*) algorithm [1], prior to describing our adaption of PILCO in Section III. PILCO is a model-based policy search RL algorithm that achieved unprecedented data-efficiency of several control benchmarks including the cartpole swing-up task.

PILCO is summarised by Algorithm 1. A policy $\pi$'s functional form is chosen by the user (step 1), whose parameters $\psi$ are initialised randomly (step 2). Thereafter PILCO executes the current policy from an initial state (sampled from initial distribution $p(X_0)$) until the time horizon $T$ (defined as one trial, step 4). Observed transitions are recorded, and appended to the total training data. Given the additional training data, the dynamics model is re-trained (step 5). Using its probabilistic transition model, PILCO then analytically predicts states distributions from an initial state distribution $p(X_0)$ to $p(X_1)$ etc. until time horizon $p(X_T)$ making a joint Gaussian assumption (step 6). Prediction of future state distribution follows the generative model seen in Figure 1, where each system-state $X_t$ defines an action $U_t$ according to policy $\pi$, which determines the new state $X_{t+1}$ according to the dynamics $f$. I.e. $X_{t+1} = f(X_t, U_t)$, where we train a GP model of $f$ given all previous observed transition tuples $\{X_t, U_t, X_{t+1}\}$. Given the multi-state distribution $p(\{X_0, ..., X_T\})$, the expected cost $\mathbb{E}_X[\text{cost}(X_t)]$ is computed for each state distribution, using a user-supplied cost function. The sum of expected costs is our minimisation objective $J$ (step 7). Gradient information is also computed w.r.t. policy parameters $\mathrm{d}J/\mathrm{d}\psi$. Finally, the objective $J$ is optimised using gradient decent according to $\mathrm{d}J/\mathrm{d}\psi$ (step 8). The algorithm then loops back to step 4 and executes the newly-optimised policy, which is locally optimal given all the data observed thus far.

**Algorithm 1** PILCO

```
1: Define policy's functional form: $\pi : z_t \times \psi \rightarrow u_t$.
2: Initialise policy parameters $\psi$ randomly.
3: repeat
4:     Execute system, record data.
5:     Learn dynamics model.
6:     Predict system trajectories from $p(X_0)$ to $p(X_T)$.
7:     Evaluate policy:
           $J(\psi) = \sum_{t=0}^{T} \gamma^t \mathbb{E}_X[\text{cost}(X_t)|\psi]$.
8:     Optimise policy:
           $\psi \leftarrow \arg\min_{\psi} J(\psi)$.
9: until policy parameters $\psi$ converge
```

PILCO's data-efficiency success can be attributed to its *probabilistic* dynamics model. Probabilistic models help avoid model bias – a problem that arises from selecting only a single dynamics model $\hat{f}$ from a large possible set, and assuming that $\hat{f}$ is the correct model with certainty [2]. Whilst such approaches can provide accurate short term state predictions e.g. $p(X_1)$, their longterm predictions (e.g. $p(X_T)$) are inaccurate due to the compounding effects of $T$-many prediction errors from $\hat{f}$. Since inaccurate predictions of $p(X_T)$ are made with high-confidence, changes in policy parameters $\psi$ are (falsely) predicted to have significant affect on the expected cost at time $T$. Since optimising total expected cost $J$ must balance the expected costs of states, including $p(X_1)$ and $p(X_T)$, the optimisation will compromise on the cost of $p(X_1)$ based on perceived cost of $p(X_T)$ – even though the prediction $p(X_T)$ is effectively random noise. I.e. given sufficient model uncertainty, $p(X_T)$ will have a broad distribution almost invariant to policy $\pi$. Such undesirable behaviour hampers data efficiency. Optimising data-efficiency exacerbates the negative effects of model bias even further, since the smaller the data, the larger the set of plausible models that can describe that data. PILCO uses probabilistic models to avoid model bias by considering *all* plausible dynamics models in prediction of all future states. In cases as the above, PILCO optimises the policy based only on the states $X_t$ it can predict well.

## III. Our Algorithm: Deep PILCO

We now describe our method – *Deep PILCO* – for data-efficient deep RL. Our method is similar to PILCO: both methods follow Algorithm 1. The main difference of Deep PILCO is its dynamics model. PILCO uses a Gaussian process which can model the dynamics' output uncertainty, but cannot scale to high dimensional observation spaces. In contrast, Deep PILCO uses a deep neural network capable of scaling to high dimensional observations spaces. Like PILCO, our policy-search algorithm alternates between

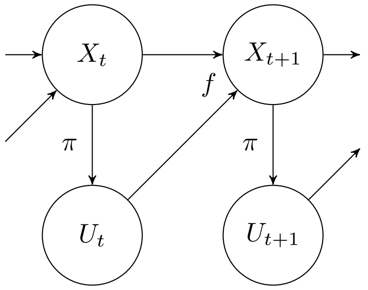


Fig. 1: **Prediction model of system trajectories** (step 6 in Algorithm 1). The system state $X_t$ generates action $U_t$ according to policy $\pi$, both of which result in a new state $X_{t+1}$ as predicted by dynamics model $f$ (a model trained given all previously observed $X$ and $U$).

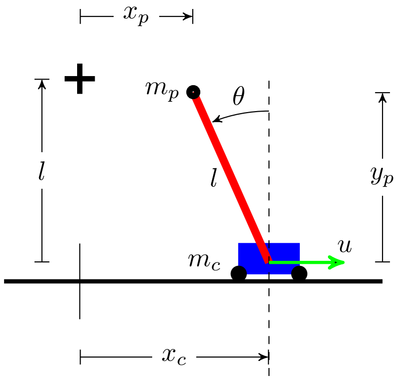


Fig. 2: **The cartpole swing-up task.** A pendulum of length $l$ is attached to a cart by a frictionless pivot. The system begins with cart at position $x_c = 0$ and pendulum hanging down: $\theta = \pi$. The goal is to accelerate the cart by applying horizontal force $u_t$ at each timestep $t$ to invert then stabilise the pendulum's endpoint at the goal (black cross).

fitting a dynamics model to observed transitions data, evaluating the policy using dynamics model predictions of future states and costs, and then improving the policy.

Replacing PILCO's GP with a deep network is a surprisingly complicated endeavour though, as we wish our dynamics model to maintain its probabilistic nature, capturing 1) output uncertainty, and 2) input uncertainty.

### A. *Output uncertainty*

First, we require output uncertainty from our dynamics model, critical to PILCO's data-efficiency. Yet simple NN models cannot express output model uncertainty, and thus cannot capture our ignorance of the latent system dynamics. To solve this we use the Bayesian probabilistic equivalent of the NN – the Bayesian neural network (BNN) [10].

In low data settings, BNNs represent model uncertainty with the use of a posterior distribution over the weights of the NN. However, the true posterior of a BNN is intractably complex. One approximate solution is to use variational inference where we find a distribution in a tractable family which minimises the Kullback-Leibler (KL) divergence to the true posterior. [6] show that dropout can be interpreted as a variational Bayesian approximation, where the approximating distribution is a mixture of two Gaussians with small variances and the mean of one of the Gaussians fixed at zero. The uncertainty in the weights induces prediction uncertainty by marginalising over the approximate posterior using Monte Carlo integration. This amounts to the regular dropout procedure only with dropout also applied at test time, giving us output uncertainty from our dynamics model.

This approach also offers insights into the use of NNs with small data. [6] for example show that the network's weight decay can be parametrised as a function of dataset size, dropout probability, and observation noise. Together with adaptive learning-rate optimisation techniques, the number of parameters requiring tuning becomes negligible.

### B. *Input uncertainty*

A second difficulty with NN dynamics models is handling *input* uncertainty. To plan under dynamics uncertainty, PILCO analytically propagates state distributions through the dynamics model (step 6 in Algorithm 1, depicted in Figure 1). To do so, the dynamics model must pass uncertain dynamics outputs from a given time step as uncertain input into the dynamics model in the next time step. This handling of input uncertainty cannot be done analytically with NNs, as is done with Gaussian processes in PILCO.

To feed a distribution into the dynamics model, we resort to particle methods (Algorithm 2). This involves sampling a set of particles from the input distribution (step 2 in Algorithm 2), passing these particles through the BNN dynamics model (and sampling the uncertain output, step 8 in Algorithm 2), which yields an output distribution of particles.

This approach was attempted unsuccessfully in the past with PILCO [7]. [7] encountered several problems optimising the policy with particle methods, the main problem being the abundance of local optima in the optimisation surface, impeding their BFGS optimisation method. [7] suggested that this might be due to the finite number of particles used and their deterministic optimisation. To avoid these issues, we randomly re-sample a new set of particles at each optimisation step, giving us an unbiased estimator for the objective (step 7 in Algorithm 1). We then use the stochastic optimisation procedure Adam [11] instead of BFGS.

We found that fitting a Gaussian distribution to the output state distribution at each time step, as PILCO does, is of crucial importance (steps 10-11 in Algorithm 2). This moment matching avoids multi-modality in the dynamics model. Fitting a multi-modal distribution with a (wide) Gaussian causes the objective to average over the many high-cost states the Gaussian spans [2]. By forcing a unimodal fit, the algorithm penalises policies that cause the predictive states to bifurcate, often a precursor to a loss of control. This can alternatively be seen as smoothing the gradients

**Algorithm 2** Step 6 of Algorithm 1: *Predict* system trajectories from $p(X_0)$ to $p(X_T)$

```
1:  Define time horizon $T$.
2:  Initialise set of $K$ particles $x_0^k \sim P(X_0)$.
3:  for $k = 1$ to $K$ do
4:      Sample BNN dynamics model weights $W^k$.
5:  end for
6:  for time $t = 1$ to $T$ do
7:      for each particle $x_t^1$ to $x_t^K$ do
8:          Evaluate BNN with weights $W^k$ and input particle
            $x_t^k$, obtain output $y_t^k$.
9:      end for
10:     Calculate mean $\mu_t$ and standard deviation $\sigma_t^2$ of
        $\{y_t^1, ..., y_t^K\}$.
11:     Sample set of $K$ particles $x_{t+1}^k \sim \mathcal{N}(\mu_t, \sigma_t^2)$.
12: end for
```

of the expected cost when bifurcation happens, simplifying controller optimisation (this is explained further, with examples, in the experiments section). We hypothesised this to be an important modelling choice done in PILCO and assessed this assumption in our experiments.

### C. Sampling functions from the dynamics model

Unlike PILCO, our approach allows sampling individual functions from the dynamics model and following a single function throughout an entire trial. This is because a repeated application of the BNN dynamics model above can be seen as a simple Bayesian recurrent neural network (RNN, where an input is only given at the first time step). Approximate inference in the Bayesian RNN is done by sampling function weights once for the dynamics model, and using the same weights at all timesteps (steps 4 and 8 in Algorithm 2). With dropout, this is done by sampling and fixing the dropout mask for all time steps during the rollout [12]. PILCO does not consider such temporal correlation in model uncertainty between successive state transitions, which results in PILCO underestimating state uncertainty at future timesteps [2].

Another consequence of viewing our dynamics model as a Bayesian RNN is that the model could be easily extended to more interesting RNNs such as Bayesian LSTMs, capturing long-term dependencies between states. This is important for non-Markovian system dynamics, which can arise with observation noise for example. In this paper we restrict the model to Markovian system dynamics, where a simple Bayesian recurrent neural network model suffices to predict a single output state given a single input state.

Figure 5 (explained in detail below) shows trials obtained with a fixed controller and sampled dynamics model functions for the cartpole swing-up task. These are generated by sampling particles from the initial distribution, and sampling and fixing a dropout mask throughout the trial for each particle.

## IV. Experiment Setup

This section describes the experiment setup we used to compare our Deep PILCO algorithm against the state-of-the-art PILCO, using the cartpole swing-up task as a benchmark. We present a detailed analysis of the experiment results in Section V. The cartpole swing-up task (Figure 2) is a standard benchmark for nonlinear control due to the nonlinearity in the dynamics, and the requirement for nonlinear controllers to successfully swing up and balance the pendulum. Apart from the cartpole swing-up task, we evaluated our approach on the cart, cartpole balancing, and pole swing-up tasks. We report results on the cartpole swing-up alone due to space constraints (and since this is the most difficult of the tasks).

The cartpole swing-up is a continuous state, continuous action, discrete time task. We assume zero observation noise. The task goal is to balance the pendulum upright. A system state $x$ comprises the cart position, pendulum angle, and their time derivatives $x = [x_c, \theta, \dot{x}_c, \dot{\theta}]^\top$. Task parameters used are pendulum length $l = 0.6$m, cart mass $m_c = 0.5$kg, pendulum mass $m_p = 0.5$kg, time horizon $T = 2.5$s, time discretisation $\Delta t = 0.1$s, and acceleration due to gravity $g = 9.82\text{m/s}^2$. In addition, friction resists the cart's motion with a damping coefficient $b = 0.1$Ns/m. The cartpole's motion is described with the differential equation:

$$\dot{x} = \Bigg[\dot{x_c},\, \dot{\theta},\, \frac{-2m_p l\dot{\theta}^2 s + 3m_p g s c + 4u - 4b\dot{x_c}}{4(m_c + m_p) - 3m_p c^2},$$
$$\frac{-3m_p l\dot{\theta}^2 sc + 6(m_c + m_p)gs + 6(u - b\dot{x_c})c}{4l(m_c + m_p) - 3m_p l c^2}\Bigg],$$

using shorthand $s = \sin\theta$ and $c = \cos\theta$. Both the initial latent state and initial belief are assumed to be i.i.d.: $X_0 \overset{iid}{\sim} \mathcal{N}(\mu, \Sigma)$ where $\mu \sim \delta([0, \pi, 0, 0]^\top)$ and $\Sigma^{\frac{1}{2}} = \text{diag}([0.2\text{m}, 0.2\text{rad}, 0.2\text{m/s}, 0.2\text{rad/s}])$.

We use a saturating cost function: $1 - \exp\left(-\frac{1}{2}d^2/\sigma_c^2\right)$ where $\sigma_c = 0.25m$ and $d^2$ is the squared Euclidean distance between the pendulum's end point $(x_p, y_p)$ and its goal $(0, l)$. This is the standard setting used with PILCO as well.

For our deep dynamics model we experimented with dropout probabilities $p = 0, 0.05, 0.1$, and $0.2$. We found that $p = 0.05$ performed best and used this in our comparison to PILCO. As per Algorithm 1 we alternate between fitting a dynamics model and optimising the policy. To fit the dynamics model we use $5 \times 10^3$ optimisation steps, each step using 100 particles (batch size). To optimise the controller we use $10^3$ steps, each step with batch size of 10. Weight decay of the NN dynamics model is set to $10^{-4}$. As we assume no observation noise in the experiment, we set the dynamics model's observation noise to $10^{-3}$. The dynamics model architecture has 200 units with 2 hidden layers and sigmoid activation functions. Our controller is a radial basis function (RBF) network of 50 units. Like [8], we use a "replay buffer" of finite size (the most recent 10 trials), discarding older trials of data.

In our experiment we generate a single random trial for each method, and then iterate Algorithm 1's main loop for 100 times (40 for PILCO due to its time complexity). At each iteration a single trial of data is acquired by executing the cartpole for 2.5s, generating 25 transition datum per trial. We evaluate each method at each iteration by calculating the controller's cost averaged over 50 randomly sampled initial states from $p(X_0)$.

Figure 3 shows the average cost of 4 random runs (with two standard deviations denoted by two shades for each plot). Deep PILCO matches PILCO's cost very quickly (within 26 trials) and then improves on PILCO's performance by 25%, converging to 0.3 cost (compared to PILCO's 0.4 cost). Figure 4 shows the progression of deep PILCO's fitting as more data is collected.

Our model can be seen as a Bayesian approach to data efficient deep RL. We compare to recent deep RL algorithms ( [8] and [9]). [8] use an actor-critique model-free algorithm based on deterministic policy gradient. [9] train a continuous version of model-free deep Q-learning using imagined trials generated with a learnt model. For their *low dimensional* cartpole swing-up task [8] require approximately $2.5 \times 10^5$

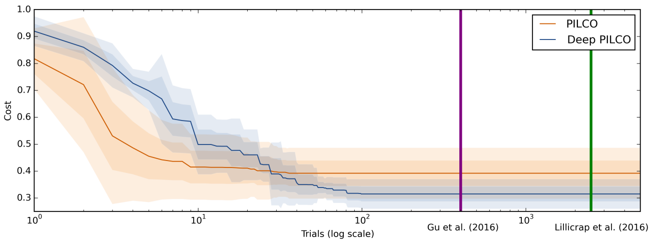


Fig. 3: Cost per trial (on *log scale*) for PILCO and Deep PILCO for the cartpole swing-up task. Vertical lines show estimates of number of trials required for model convergence for [9] (purple, requiring $\sim$ 400 trials) and [8] (green, requiring $\sim 2,500$ trials).

steps to achieve good results. This is equivalent to approximately $2.5 \times 10^3$ trials of data, based on Figure 2 in [8] (note that [8] used time horizon $T = 2$s and time discretisation $\Delta t = 0.02$s, slightly different from ours; they also normalised their reward, which does not allow us to compare to their converged reward directly). [9] require approximately 400 trials for model convergence. These two results are denoted with vertical lines in Figure 3 (as the respective papers do not provide high resolution trial-cost plots).

Lastly, we report model run time for both Deep PILCO as well as PILCO. Deep PILCO can leverage GPU architecture, and took 5.85 hours to run for the first 40 iterations. This is with constant time complexity w.r.t. the number of trials, and linear time complexity in input dimensionality $Q$ and output dimensionality $D$. PILCO (running on the CPU) took 20.7 hours for the 40 trials, and scales with $\mathcal{O}(N^2Q^2D^2)$ time complexity, with $N$ number of trials. With more trials PILCO will become much slower to run. Consequently, PILCO is unsuited for tasks requiring a large number of trials or high-dimensional state tasks.

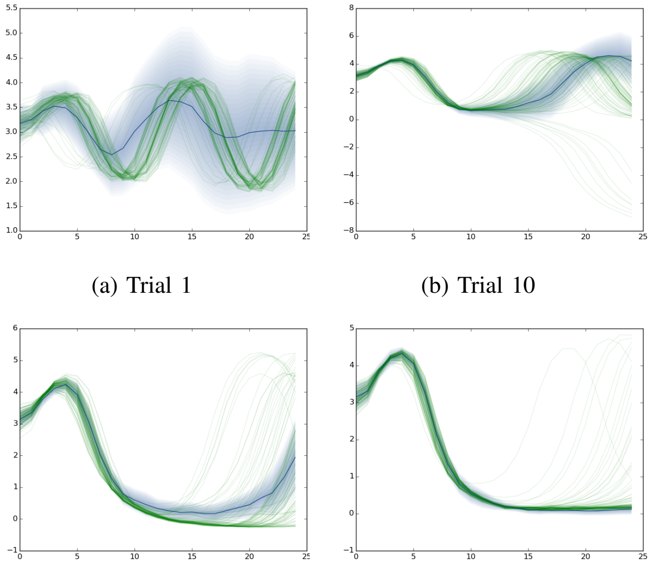
## V. Analysis

We next analyse the results from the previous section, providing insights based on the different setups we experimented with.

### A. *Dynamics Models of Finite Capacity*

PILCO's dynamics model is a Gaussian process, a non-parametric model of infinite-capacity, effectively able to memorises all observed transition data. As more data is observed, the model's flexibility increases to fit the additional data. NNs on the other hand are of finite capacity, and must "smooth" over different data points when fitting to the data. This poses a difficulty in our setting since when fitting a NN model to a sequence of trials, the same importance would be given to new trials as old ones. But new trials are much more important for the agent to learn, and with many old trials a NN might not model the new ones well.

One possible solution would be to use a larger NN with each iteration, making inference slower as we get more data. Another solution is to downweigh older trials, and use a weighted Euclidean loss (which can be seen as a naive form of "experience replay"). We experimented with an exponential decay, such that data from the oldest trial either had weight 0.01, 0.1 or 0.4. A disadvantage of this approach is slower optimisation. It requires several passes over the training dataset with most observations being inconsequential to the optimisation and did not work well in practice for any decay rate. We suspect this to be due to the finite capacity of the NN, where the function complexity keeps increasing but the model complexity does not.



(a) Trial 1

(b) Trial 10

(c) Trial 20

(d) Trial 26

Fig. 4: **Progression of model fitting and controller optimisation as more trials of data are collected.** Each x-axis is timestep $t$, and each y-axis is the pendulum angle $\theta$ in radians (see Figure 2). The goal is to swing the pendulum up such that $\text{mod}(\theta, 2\pi) \approx 0$. The green lines are samples from the ground truth dynamics. The blue distribution is our Gaussian-fitted predictive distribution of states at each timestep. **(a)** After the first trial the model fit (blue) does not yet have enough data to accurately capture the true dynamics (green). Thus the policy performs poorly: the pendulum remains downwards swinging between $2\pi$ and $4\pi$. **(b)** After 10 trials, the the model fit (blue) predicts very well for the fist 13 timesteps before separating from the true rollouts (green). The controller has stabilised the pendulum at $0\pi$ for about 10 timesteps (1 second). **(c)** After 20 trials the model fit and policy are slightly improved. **(d)** From trial 26 onward, the dynamics model successfully captured the true dynamics and the policy successfully stabilises the pendulum upright at $0\pi$ radians most trials.

Our solution was instead to keep the last 10 trials of data only, similar to [8]'s "replay buffer". A disadvantage of this approach is that the agent might forget bad trials, and will continuously attempt to explore these even with an good controller.

### B. *Particle Methods and Moment Matching*

In our method we moment-matched the output distribution at each time step before propagating it to the next time step.

This forces the state distribution to be uni-modal, avoiding bifurcation points. We hypothesised in Section III that this is an important modelling choice in PILCO. To assess this hypothesis, we experimented with an identical experiment setup to the above, but without moment matching. Instead, we pass the particles unchanged from the output of one time step to the next.

Figure 5 shows the dynamics model fit (in a similar plot to Figure 4). The agent is able to swing the pendulum up and maintain it for 0.5s, but then loses control and has to swing it up again. The state trajectories seem to bifurcate at the origin, with half of the states swinging clockwise, and half counter-clockwise. The controller seems to be located at a local optimum, as the agent seems unable to escape this behaviour within 100 iterations (either with the 10 trials memory restriction, or without).

Imposing a moment matching modelling assumption, the agent would incur much higher cost at time step 9 for example. Instead of having half of the trajectories at $2\pi$ and the other half at $0\pi$ (resulting in overall low cost), a Gaussian fit to these trajectories will have mean $\pi$ and a large standard deviation. Sampling new particles from this Gaussian will have almost all particles with a high cost. More importantly though, these particles will have near-zero gradients (using the exponential cost function). This allows the controller optimiser to ignore these states, and go in the direction of gradients from states with low cost that do not bifurcate.

### C. Importance of Being Uncertain About Model Dynamics

We assessed the importance of a probabilistic dynamics model in our setting. This can easily be done by setting the dropout probability to $p = 0$, which results in a MAP estimate. As can be seen in Figure 6a, even after 75 trials the dynamics model would not converge to anything sensible. This suggests that input uncertainty is not sufficient, and propagating the input distribution with a MAP estimate will not necessarily avoid model bias. The same behaviour is observed with too high dropout probability (Figure 6d) presumably as the model underfits. Dropout probabilities of 0.05 and 0.1 seem to work best (Figures 6b and 6c). Note though that these values are dependent on model size, and with larger NNs we would expect larger dropout probabilities to work better.

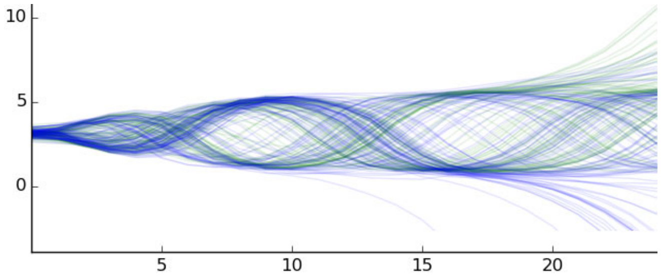
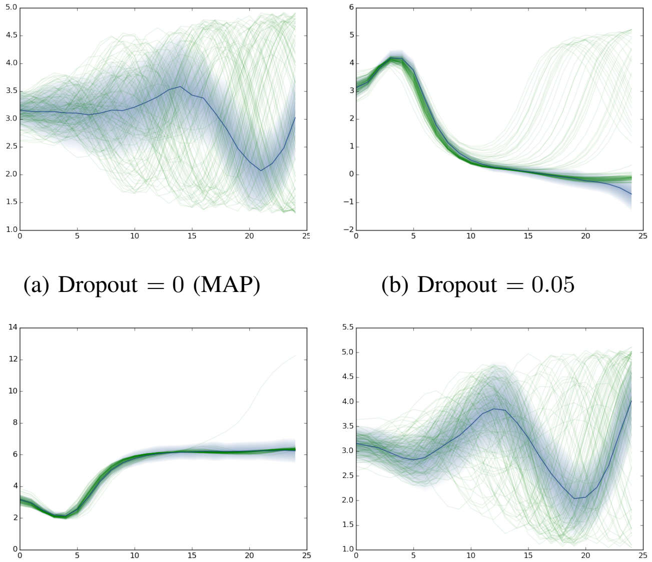


Fig. 5: **Pendulum angle $\theta$ (from Figure 2) as a function of timesteps.** Each trajectory corresponds to a single particle sampled from the initial distribution at $t = 0$. A controller is optimised using Deep PILCO and fixed. Blue trajectories are pendulum angle following the learnt dynamics model with the fixed controller, and green trajectories are pendulum angle following the system (true) dynamics with the same controller. Each blue trajectory follows a single sampled function from the dynamics model (applying the same dropout mask at each timestep). The learnt dynamics model matches the system dynamics, and the particle distribution successfully captures the multi-modal predictive distribution of future states. However, without moment matching we could not optimise a good controller from these multi-modal distributions.



(a) Dropout $= 0$ (MAP)

(b) Dropout $= 0.05$

(c) Dropout $= 0.1$

(d) Dropout $= 0.2$

Fig. 6: **Effects of dropout probabilities on the dynamics model after 75 trials.** Each x-axis is timestep $t$, and each y-axis is pendulum angle $\theta$. MAP estimate fails to capture the dynamics as it offers no probabilistic output (the depicted uncertainty is that of the propagated input distribution). Too high dropout probability (0.2) does not allow the model to learn the system dynamics.

## VI. Future work

An exciting set of options exist for future work. First, we can easily extend the dynamics model to consider observation noise – even heteroscedastic observation noise [13]. For example, heteroscedastic observation noise exists in using a camera to observe the position of moving objects. The speed of the pendulum increases uncertainty in observing the pendulum's position due to blurring of the image. Capturing observation noise can be challenging with the standard PILCO framework, but would be a simple extension in ours. Second, we can consider observation noise and incorporate a filter, analogous to [14]. Third, several compression techniques remain available to learn in high dimensional spaces (e.g. pixels) by compressing down to low dimensional state representations, avoiding the use of data-inefficient autoencoders. Fourth, we can increase gradient smoothness to better facilitate the policy optimisation process by smoothing the particles at each point with Gaussian bumps. Lastly, we can increase data-efficiency using exploration. Since we use probabilistic models throughout planning we can use uncertainty-directed exploration which is much more efficient and informed than undirected random exploration techniques such as epsilon-greedy or Boltzmann distributions [15].

## REFERENCES

[1] M. Deisenroth and C. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[2] M. Deisenroth, D. Fox, and C. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 37, no. 2, pp. 408–423, 2015.

[3] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[4] W. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[5] C. Atkeson and J. Santamaria, "A comparison of direct and model-based reinforcement learning," in *In International Conference on Robotics and Automation*. Citeseer, 1997.

[6] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," *arXiv preprint arXiv:1506.02142*, 2015.

[7] A. McHutchon, "Nonlinear modelling and control using gaussian processes," Ph.D. dissertation, University of Cambridge, 2014.

[8] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[9] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," *arXiv preprint arXiv:1603.00748*, 2016.

[10] D. MacKay, "Bayesian methods for adaptive models," Ph.D. dissertation, California Institute of Technology, 1992.

[11] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[12] Y. Gal, "A theoretically grounded application of dropout in recurrent neural networks," *arXiv:1512.05287*, 2015.

[13] ——, "Homoscedastic and heteroscedastic models," *https://github.com/yaringal/ HeteroscedasticDropoutUncertainty*, 2016.

[14] R. McAllister and C. Rasmussen, "Data-efficient reinforcement learning in continuous-state POMDPs," *arXiv preprint arXiv:1602.02523*, 2016.

[15] S. Thrun, "Efficient exploration in reinforcement learning," 1992.