Carl Edward Rasmussen and Marc Peter Deisenroth

Probabilistic Inference for Fast Learning in Control

Chapter in *Recent Advances in Reinforcement Learning,*

Lecture Notes in Computer Science, vol. 5323, pp. 229–242, November 2008.

The original publication is available at

http://www.springer.com/computer/artificial/book/978-3-540-89721-7.

Probabilistic Inference for Fast Learning in Control

Carl Edward Rasmussen[1,2] and Marc Peter Deisenroth[1,3]

[1] Department of Engineering, University of Cambridge, UK
[2] Max Planck Institute for Biological Cybernetics, Tübingen, Germany
[3] Faculty of Informatics, Universität Karlsruhe (TH), Germany

Abstract. We provide a novel framework for very fast model-based reinforcement learning in continuous state and action spaces. The framework requires probabilistic models that explicitly characterize their levels of confidence. Within this framework, we use flexible, non-parametric models to describe the world based on previously collected experience. We demonstrate learning on the cart-pole problem in a setting where we provide very limited prior knowledge about the task. Learning progresses rapidly, and a good policy is found after only a hand-full of iterations.

1 Introduction

Learning from experience is a key ingredient in the behavior of intelligent beings and holds great potential for artificial systems. Algorithms for learning from experience are studied in the areas of reinforcement learning (RL) and adaptive control. A central issue for such algorithms is the *speed of learning*, that is, the number of trials necessary to learn a task. Many learning algorithms require a huge number of trials to succeed, whereas biological systems often learn quickly.

There are broadly two types of approaches to speed up learning of artificial systems. One approach is to constrain the task in various ways to simplify learning. The issue with this approach is that it is highly problem dependent and relies on an a priori understanding of the characteristics of the task. Alternatively, one can speed up learning by extracting more useful information from available experience. This effect can be achieved by carefully modeling the observations. In a practical application, one would typically combine these two approaches. In this paper, we are concerned solely with the second approach: How can we learn as fast as possible, given only very limited prior understanding of a task? Thus, we are not looking for an engineering solution to a particular problem, but rather we elicit a general algorithm for effective learning. The approach is general. For purposes of illustration, we will apply it to the well-known cart-pole problem.

1.1 Related Work

Experience from real interactions can be used for two purposes. It can be used either to update the current model of the world (indirect RL) or it can be used to improve the value function and/or policy directly (direct RL), or combinations of the two. With a learned model, it is possible to get *simulated* experience to perform a planning update resulting in a new (model-specific) policy and/or value function. Real experience is used to build the model and not to directly optimize the policy itself. This idea is described by Sutton's Dyna architecture introduced in [1].

Direct and indirect methods have different strengths and weaknesses. On the one hand, model-free algorithms do not rely on a (possibly incorrect) model. On the other hand, they require many interactions with the real system to find a solution to the considered RL problem. In real-world problems, hundreds of thousands or millions of interactions with the real system are often infeasible due to time, physical, and monetary constraints. In contrast to model-free methods, indirect (model-based) approaches can make more efficient use of limited experience. However, they may suffer if the model employed is not a sufficiently good approximation to the real world. This problem was already recognized by Atkeson and Santamaría [2] and Atkeson and Schaal [3]. To overcome the problem of policies for inaccurate models, Abbeel et al. add a heuristic bias term when updating the model after gathering real experience, [4]. In [5], Poupart et al. learn a probabilistic model for a finite state POMDP problem to incorporate observations into prior knowledge. However, a principled and rigorous way of building models that consistently quantify knowledge and uncertainty does not exist in the RL literature to our best knowledge. In our approach, we use flexible non-parametric probabilistic models to reap the benefit of the indirect approach while minimizing any problems of model bias.

Traditionally, solving even relatively simple tasks from scratch have been considered "daunting", [6], in the absence of strong task-specific prior assumptions. In the context of robotics, one popular solution employs prior knowledge provided by a human "teacher" to restrict the solution space [3,4,6,7,8]. The teacher shows the robot what a possible solution looks like by demonstrating it. Building a (local) model using data from this demonstration might also alleviate model errors along a good trajectory.

In contrast to previous work, we propose a fast RL algorithm which is able to learn a good policy *without* problem-specific prior knowledge. Similar to [9] and [10], we consider model-based policy iteration using probabilistic dynamics models. However, in [9,10], actions are considered as parameters to be optimized in any state rather than modeling the policy as a function of the state. In contrast to [9] and [10], we propose learning the policy explicitly.

2 Fast Reinforcement Learning Framework

For fast reinforcement learning, we propose an adaptive probabilistic world model learned on previous experience. Due to limited experience, a probabilistic model is required to appropriately model what is known and what not. The model is used in a planning step to determine a good (model-optimized) policy.

Algorithm 1 Fast reinforcement learning

```
1: initial exploration                                        ▷ interaction phase
2: loop
3:     collect observations
4:     update probabilistic dynamics model
5:     optimize policy through simulation                      ▷ planning phase
6:     apply (model-optimized) policy to real system           ▷ interaction phase
7: end loop
```

A high-level description of the general framework is given in Algorithm 1. We distinguish between the phase of real interaction and simulation in the planning phase. The algorithm is initialized by an arbitrary policy and some notion about the current state of the world. In the interaction phase, we take this information and apply a single action to the real world according to our current policy. We observe a new state and employ the policy again. Subsequently, the probabilistic world model is updated using new and historical experience collected during the interaction phases. During the planning phase, we take a state distribution and *simulate* the world with the corresponding distribution over actions. The probabilistic world model determines a distribution over successor states. The controller computes the corresponding distribution over costs and the system is being simulated by applying the policy to the entire state distribution.

During learning, the policy will be optimized in the planning phase based on the evidence so far. However, it may be that because of limited experience the simulations do not correspond to the real world. When this situation occurs and we apply the model-optimized policy to the real system, the model will discover the discrepancy between the model's predictions and the world. This new insight will be incorporated into the subsequent model update.

Crucially, in order for the internal simulations to reflect the real world as accurately as possible, the dynamics models must faithfully represent their fidelities of how accurate they are. For example, if a state is visited on a simulated trajectory about which not much knowledge has been acquired, the model must be able to quantify this uncertainty, and not simply assume that its best guess is close to the truth. A probabilistic model quantifies knowledge and can be considered as a model that captures all plausible models in a distribution over models. The use of probabilistic models for the dynamics allows us to keep track of the uncertainties in the simulations used for planning. Typically, in early stages of learning, the uncertainty in the states will grow with increasing prediction horizon. When applying a good real-world policy on the other hand, we expect the uncertainty to collapse because the system is being controlled. With increasing experience, the probabilistic model will tend to a deterministic one.

Modeling a dynamic system is generally fairly easy for short time horizons, but gets progressively more challenging as the horizon increases. Therefore, we are forced to learn the dynamics model on relatively short time scales. However, good control strategies often require the consideration of long-term effects of immediate actions. To bridge this gap, we need to cascade many short-term predictions to assess long-term behavior during the planning phase.

Our approach explicitly requires a probabilistic world model in the planning step although the framework resembles Sutton's Dyna architecture introduced in [1]. However, we believe that utilizing a deterministic model is inconsistent when uncertainties are involved.

Cost Function Let us revisit Algorithm 1. In the planning stage, the policy is optimized to minimize the expected long-term cost starting from an initial state distribution. The expected immediate cost is a function of the state. Traditionally, the squared error cost function has been extensively used. However, because of its convexity and unboundedness, the cumulative cost will be highly dependent on the worst state along a trajectory. Initially, when the dynamics model is uncertain, the uncertainty may grow rapidly with the time horizon, so that the expected cost will be highly sensitive to details of a distribution which essentially encodes that the model has "lost track of the state". To avoid the extreme dependence on these essentially arbitrary details, we use instead an immediate cost function which is locally quadratic, but which saturates for large deviations from the desired goal.

Stochastic Simulation Although the policy is a deterministic function of the state, the simulation of trajectories involves distributions over trajectories. The states are uncertain since the learned dynamics model is probabilistic. Intuitively speaking, as the probabilistic model is a distribution over all plausible models, a deterministic state is being mapped through all plausible transition functions resulting in a distribution over successors states. In the simulation phase, the distribution over states thus implies a distribution over actions, even when the policy is deterministic. When the policy is applied to the real system where the current state has just been measured, a single action is applied deterministically.

3 Implementation

In the following, we describe how to implement the general ideas from the previous section in discrete-time setting with continuous states and actions.

The probabilistic short-term dynamics models are implemented using flexible non-parametric models based on Gaussian processes (GPs). State uncertainty is explicitly propagated forward to obtain a probabilistic representation of long-term behavior. These computations can be done approximately in closed form, and Markov chain Monte Carlo is not necessary. Conditioned on the probabilistic dynamics model, the policy is optimized using policy iteration. Since the implicit internal simulation of the system can be done analytically, we have a computationally efficient method to perform a policy evaluation step. Moreover, we can compute the gradient of the expected long-term cost with respect to the policy parameters analytically, which allows the use of standard optimization methods, such as conjugate gradients.

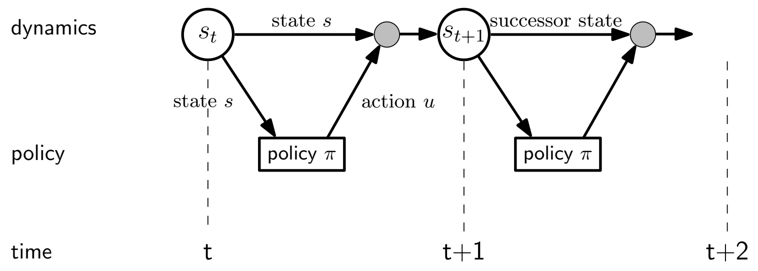

Fig. 1. Interplay of dynamics GP and policy to propagate uncertainty over time. Consider a state distribution $p(\mathbf{s}_t)$ at time step t. The policy takes this distribution as input and returns a distribution over actions. The dynamics model takes a fully joint Gaussian of states and actions (shaded nodes in the figure) as input distribution and determines mean and covariance of the successor state distribution $p(\mathbf{s}_{t+1})$ analytically as detailed in [12]. The distribution $p(\mathbf{s}_{t+1})$ is approximated by a Gaussian with the corresponding mean and covariance.

3.1 Dynamics Model

We propose learning the dynamics using Gaussian process models. A GP is a distribution over functions and utilized for state-of-the-art Bayesian non-parametric regression. Regarding a function as an infinitely long vector, all necessary computations can be broken down to manipulating standard Gaussian distributions. Thus, GP regression combines both flexible non-parametric modeling and tractable Bayesian inference. For further details, please refer to [11].

The GP dynamics model takes as input a representation of the current state and action. As output the GP computes a representation of the distribution over consecutive states. In particular, for a D-dimensional state, we utilize D separate GPs, one for each state dimension, explicitly incorporating correlations between state variables. The dynamics models are learned using the observed state trajectories by the standard algorithm (evidence maximization), see [11].

In the planning stage, the predicted state trajectories are uncertain. We therefore need to be able to predict outputs when the inputs are uncertain. To carry out the necessary computations, we use results from Girard et al., [12], and Kuss [9]. Generally, a Gaussian state followed by a nonlinear dynamics results in a non-Gaussian successor state. We follow the above references and compute exactly the two first moments of the resulting distribution, that is, a Gaussian approximation. By iteration, we can thus compute the distribution over trajectories in closed form.

Throughout all computations, we explicitly take the uncertainty about the dynamics into account by averaging over all plausible dynamics models given the current set of observations from the real world. The variances of the predicted successor states take into account both the uncertainty in the current state *and* the possibly imprecise model of the actual dynamics.

3.2 Policy Model

The controller implements a nonlinear deterministic policy. While any function approximator can be used for this purpose, we apply (the mean of) a non-parametric Gaussian process policy model. This policy GP is parameterized by a (pseudo-) training set, consisting of pairs of states (training inputs) and corresponding actions (training targets). By modifying this training set, we can control the policy being implemented. This idea is related to the sparse Gaussian process approximation using inducing inputs introduced by Snelson and Ghahramani in [13]. In contrast to [13], we do not average over the training targets, but optimize them as well. Note that since the policy GP predicts deterministically, the uncertainty about the underlying policy is zero.

In this paper, we consider deterministic policies only: For a deterministic input, the policy always returns the same control. However, due to the probabilistic dynamics model there will be uncertainties about states resulting in distributions over actions as described in Section 3.1. As illustrated in Figure 1, during the planning phase, we cascade short-term dynamics models to predict what is going to happen in the long term. When interacting with the real system, we assume that the state is measured deterministically. Then, the controller applies the (unique) corresponding control signal. In other words, we apply only a single action deterministically when interacting with the real system, but we consider a distribution over actions in simulation.

3.3 Policy Iteration

Determination of an optimal policy through policy iteration is a natural choice within the proposed framework. In the following, we will show that the policy can be evaluated analytically. Moreover, we provide a framework where the gradient of the expected cumulative cost with respect to the policy parameters can be determined analytically.

Policy Evaluation Assume a given policy π and a given probabilistic dynamics model. To evaluate the quality of the policy starting from a particular state distribution $p(\mathbf{s}_0)$, the expected cost of the trajectory $\boldsymbol{\tau} := (\mathbf{s}_0, \pi(\mathbf{s}_0), \ldots, \mathbf{s}_T)$ has to be determined. Assuming time-additive losses and a Markovian structure of the problem, the expected undiscounted finite-horizon cost

$$V^\pi(\boldsymbol{\tau}) \;:=\; \mathrm{E}_{\boldsymbol{\tau}}\Big[\sum_{t=0}^{T} \ell(\mathbf{s}_t)\big|\pi, p(\mathbf{s}_0)\Big] \;=\; \sum_{t=0}^{T} \mathrm{E}[\ell(\mathbf{s}_t)|\pi, p(\mathbf{s}_0)] \tag{1}$$

has to be evaluated, where the expectation is taken with respect to the probability distribution over trajectories $\boldsymbol{\tau}$. As described in Section 3.1, a Gaussian approximation of the predictive distributions of future states can be determined analytically. To evaluate equation (1), it remains to compute $\mathrm{E}[\ell(\mathbf{s}_t)|\pi, p(\mathbf{s}_0)]$ for an immediate cost function ℓ. If we restrict ℓ for example to contain combinations involving trigonometric functions, exponentials, and powers, this integral is analytically tractable and the gradient of equation (1) with respect to the policy parameters can be computed analytically.

Policy Optimization To optimize the policy, we minimize the expected long-term cost (1) with respect to the policy parameters using a gradient-based method. Two different kinds of parameters are involved in the Gaussian process controller. Firstly, the hyper-parameters of the kernel and secondly, the pseudo training set of the controller itself. All these parameters are collected inside the parameter vector $\boldsymbol{\theta}$. They are treated as free variables to be optimized to find an optimal strategy. In contrast to standard parameter optimization for GPs by maximizing the marginal likelihood, the objective function in our RL setup is the expected long-term cost given by equation (1).

We employ an efficient conjugate gradients minimizer, which requires the partial derivatives of the objective function with respect to the policy parameters. These derivatives can be computed analytically by repeated application of the chain rule applied to the parameters of the *distributions* governing the states over time. The gradient of V^π, equation (1), with respect to the policy parameters is given by

$$\frac{\mathrm{d}V^\pi(\boldsymbol{\tau})}{\mathrm{d}\boldsymbol{\theta}} = \sum_{t=0}^{T} \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}} \,\mathrm{E}[\ell(\mathbf{s}_t)|\pi_{\boldsymbol{\theta}}, p(\mathbf{s}_0)]\,,$$

where the subscript $\boldsymbol{\theta}$ stresses the dependency of π on the parameter set $\boldsymbol{\theta}$. The total derivative with respect to the policy parameters is denoted by $\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}$. As we know the approximate (Gaussian) state distribution $p(\mathbf{s}_t)$, we just have to compute

$$\left(\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\mu}_t}\,\mathrm{E}[\ell(\mathbf{s}_t)|\pi_{\boldsymbol{\theta}}, p(\mathbf{s}_0)]\right)\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\theta}} + \left(\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\Sigma}_t}\,\mathrm{E}[\ell(\mathbf{s}_t)|\pi_{\boldsymbol{\theta}}, p(\mathbf{s}_0)]\right)\frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\boldsymbol{\theta}}$$

for $t = 0, \ldots, T$, where $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ are mean and covariance of $p(\mathbf{s}_t)$, respectively. Exploiting the Markov property, required computations boil down to

$$\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\mu}_{t-1}}\frac{\mathrm{d}\boldsymbol{\mu}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\Sigma}_{t-1}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\theta}}\,, \tag{2}$$

$$\frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial\boldsymbol{\Sigma}_t}{\partial\boldsymbol{\mu}_{t-1}}\frac{\mathrm{d}\boldsymbol{\mu}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\Sigma}_t}{\partial\boldsymbol{\Sigma}_{t-1}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\Sigma}_t}{\partial\boldsymbol{\theta}}\,, \tag{3}$$

where $\boldsymbol{\theta}$ contains all policy parameters and $\frac{\partial}{\partial\boldsymbol{\theta}}$ denotes the partial derivative with respect to the parameter vector $\boldsymbol{\theta}$. Note that the moments of the state distribution $p(\mathbf{s}_t)$ is functionally dependent on the parameter vector $\boldsymbol{\theta}$ and the moments $\boldsymbol{\mu}_{t-1}$ and $\boldsymbol{\Sigma}_{t-1}$ of the state distribution at time $t-1$. In principle, these computations are straightforward although the details are somewhat lengthy.

4 Experiments

For demonstration purposes, we apply our approach to learning a controller for the underactuated cart-pole problem. The cart-pole task is depicted in Fig-

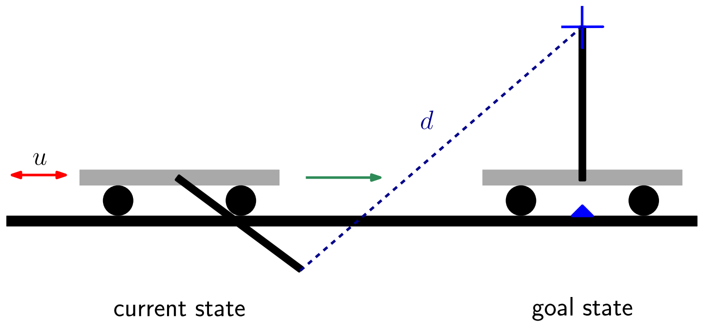

Fig. 2. Cart-pole problem. The pendulum has to be swung up and balanced at the cross by just pushing the cart to left and right. The Euclidean distance between the tip of the pendulum and the goal state is shown by the dashed line.

ure 2. As in Doya's paper [14], the pendulum has to be swung up and balanced. However, instead of just balancing the pendulum, we additionally require the pendulum to be balanced in a specific location given by the cross in Figure 2. Note that the solution of the task implies that the cart stops at the triangle. One point which makes the problem non-trivial is that to solve it, sometimes actions have to be taken which temporarily move the pendulum further away from the target. Thus, greedily optimizing a short-term cost will lead to a policy that fails to achieve the task. In control theory, the solution of the task is usually based on an intricate understanding of the system dynamics, which we do not assume in this paper. Our objective is to learn a good policy without a prior understanding of the system.

4.1 Cost Function

In general, we are interested in understanding the generic principles, which allow good solutions to be learned automatically. Hence, the only built-in assumption is that the state variables evolve smoothly over time. The state $\mathbf{s}$ of the system consists of cart position x, cart velocity $\dot{x}$, angle[4] of the pole φ, and angular velocity of the pole $\dot{\varphi}$.

The only feedback the controller gets about the quality of its applied action is the squared distance

$$d(\mathbf{s})^2 = x^2 + 2xl\sin(\varphi) + 2l^2 + 2l^2\cos(\varphi)$$

between the tip of the pendulum and its desired position, measured every 200 ms. The distance d is denoted by the dashed line in Figure 2. Note, that in contrast to common implementations, d only depends on the position variables x, $\sin(\varphi)$,

[4] Since the angle is periodic, we actually encode it in the input to the dynamics GP and policy GP as the two variables $\sin(\varphi)$ and $\cos(\varphi)$, to avoid any discontinuity in the representation.

and $\cos(\varphi)$. In particular, it does not depend on the velocity variables. We choose the immediate cost

$$\ell(\mathbf{s}) = 1 - \exp(-\tfrac{1}{2}d(\mathbf{s})^2/c^2) \in [0, 1]\,, \tag{4}$$

where $c = 0.25\,\mathrm{m}$ is a constant giving the distance at which the cost function switches between locally quadratic and saturation. The cost is zero at the goal state, increases with distance, but saturates at unity.

We generally think this cost is a better cost function than the standard quadratic since for very uncertain states we will get a cost approaching unity reflecting that the state is simply "not good", whereas the quadratic cost would depend crucially on the exact value of the error bar and highly penalize the most extreme state variable.

Our choice of cost function is supposed to be a naïve, intuitive cost which does not rely on an intricate understanding of the physical system. For example, with a little more hindsight we may have chosen to also penalize the velocity variables departure from zero. However, we consider this as a part of the challenge of the learning problem. Our choice of the cost function reflects ignorance about the dynamics as would be the case if we were using our algorithm to solve a complex novel task. Elaborate tuning of the cost function (including also control penalty) has indeed been used extensively in the literature to simplify problems. Here, we derive our cost simply from purely geometric consideration. The only additional information is the constant $c = 0.25\,\mathrm{m}$, giving a rough idea of what it means to be "close" to a solution (note for comparison that the pendulum length is $0.6\,\mathrm{m}$).

4.2 Experimental Setup

The dynamics of the cart-pole system follow the ODEs

$$\ddot{x} = \frac{\left(\frac{u - b\dot{x} + m\frac{l}{2}\dot{\varphi}^2\sin\varphi}{I + m(\frac{l}{2})^2} + m^2(\frac{l}{2})^2 g\sin\varphi\cos\varphi\right)}{(M+m)(I + m(\frac{l}{2})^2) - m^2(\frac{l}{2})^2(\cos\varphi)^2}\,,$$

$$\ddot{\varphi} = \frac{m\frac{l}{2}\cos\varphi(u - b\dot{x} + m\frac{l}{2}\dot{\varphi}^2\sin\varphi) + (M+m)gm\frac{l}{2}\sin\varphi}{m^2(\frac{l}{2})^2(\cos\varphi)^2 - (M+m)(I + m(\frac{l}{2})^2)}\,,$$

where $M = 0.5\,\mathrm{kg}$ is the mass of the cart, $m = 0.5\,\mathrm{kg}$ the mass of the pole, $b = 0.1\,\mathrm{N\,s/m}$ the friction between cart and ground, $l = 0.6\,\mathrm{m}$ the length of the pole, $I = 0.06\,\mathrm{kg\,m^2}$ the moment of inertia around the tip of the pole, and $g = 9.82\,\mathrm{m/s^2}$ the gravitational constant.

The control $u \in [-10, 10]\,\mathrm{N}$ is a horizontal force pushing the cart to left or right. To guarantee that the controller learns and implements only forces in the admissible range $[-u_{\max}, u_{\max}] = [-10, 10]\,\mathrm{N}$, the probability distribution of the control signal is squashed through the sine function, such that $p(u) = p(u_{\max}\sin(\pi(\mathbf{s})))$.

For the dynamics model, we use four separate GP models, one for each state variable. The policy is implemented by a GP, which is parameterized by a pseudo training set of pairs of states and actions. Throughout the experiments, we use a pseudo training set with 50 elements, and accordingly the policy contains approximately 300 free parameters.

The eigen-frequency of this system is of the order of 1 Hz, and we use a short-term prediction time of 0.2 seconds. Note that this is a much slower sampling time than is typically used in conventional controllers for this problem. It is adequate here, as it is easy to capture the dynamics at this time scale. We optimize the objective function (1) over 5 s, that is, 25 time steps. In our setting, the initial state distribution $p(\mathbf{s}_0)$ is Gaussian with zero mean and covariance matrix $\mathbf{\Sigma} = 10^{-4}\mathbf{I}$. The goal state is $\mathbf{s}_{\text{goal}} = [0, 0, \pi, 0]^T$. In other words, the initial state is that the cart is in the right position, but the pendulum is hanging down (instead of being balanced in the upright position). The task for the learning algorithm is thus to explore the state space and to find and to exploit a strategy which will achieve the swing-up and balancing. Note that this task is not achievable by a linear controller.

We implement policy iteration within the fast RL framework given by Algorithm 1 as follows. Initially, we assume fully unknown transition dynamics. To build a first dynamics model, we have to gather some experience. We observe two short (five seconds) trajectories of the system by applying forces randomly starting from an initial state since we do not have prior knowledge about a good policy. We initialize 50 pseudo training inputs of the controller to be the states along the random trajectories. The corresponding pseudo training targets are initialized randomly distributed around zero. In the next step, we build a probabilistic Gaussian process model of the transition dynamics using the observed data. Utilizing this model, we simulate the dynamics for five seconds and optimize the policy parameters using conjugate gradients.[5] Now, an optimized policy for the current dynamics model has been determined, and we are ready to apply the policy to the real system again. The application of the model-optimized policy is presumably not optimal when applied to the real system and, therefore, might lead the real cart-pole system to unexpected states. However, the applied policy is better than just applying random forces, such that states closer to the goal state are visited. We collect these new observations and update the dynamics model by incorporating all experience from previous interactions with the real system.

Alternating, the policy is optimized based on the dynamics model, and the model itself is updated based on collected data when applying the policy to the real system. With each iteration the probabilistic model describes the dynamics better and with more confidence.

4.3 Evaluations

In Figure 3, the predicted immediate costs and the costs incurred when applying the optimized policy to the real system are plotted over a horizon of 5 s. The system is started in the state where the pendulum is hanging down.

[5] Note that the derivatives (2) and (3) can be computed analytically exactly.

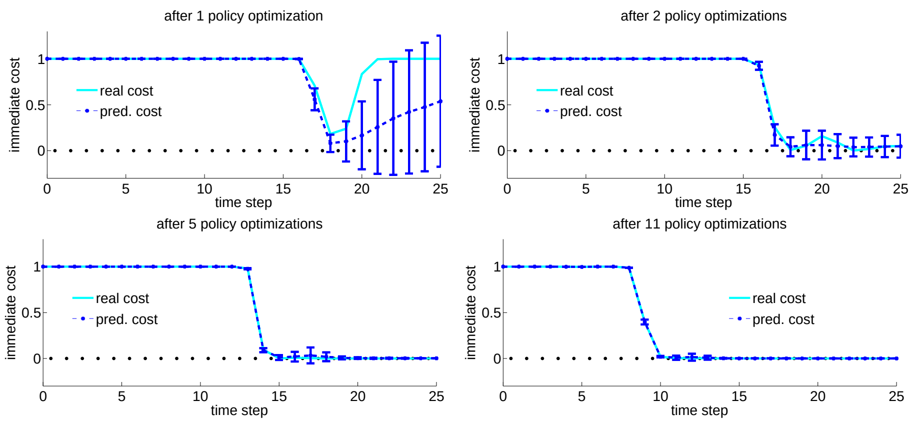

Fig. 3. Predicted costs and real costs after 1, 2, 5, and 11 policy optimizations (from top left to bottom right). The x-axis is the number of 200 ms time steps, the y-axis is the immediate costs. The black dots denote the minimum possible immediate cost when the pendulum is exactly in the goal state. The solid graphs show the real costs, the dashed graphs show the predicted immediate cost distribution. The error bars are twice the standard deviation.

In the top left plot, we see that for the first roughly 3 seconds the system does not enter states with a cost significantly different from unity. At about 3 seconds the model predicts a decline in cost, but simultaneously a rapid increase in the error bars is seen. This reflects the poor dynamics model around the goal state as we have never seen any data in this region. Applying the found policy, we see (in the solid line) that indeed the cost does decrease after about 3 seconds. Furthermore, the actual trajectory lies roughly in agreement with the model's assessment of its own accuracy. In the top left hand plot, the model now has 5 seconds more experience, also including states close to the goal state. The model now predicts much smaller error bars, and that the small cost can be maintained until the end of the simulation. The actual trajectory is in agreement with this. Iterating the learning procedure for more steps results in even smaller error bars and in a quicker swing-up action. The determined policy is not necessarily an optimal one, but it is doing a fairly good job, and the system has found it automatically in less than 60 seconds experience. See `http://mlg.eng.cam.ac.uk/carl/ewrl08` for demos of the learning system. Doya solved a similar problem requiring about 20,000 seconds (approximately 5.5 hours) of interactions with the real system to perform the swing-up reliably, [14].

Note that in the above experiments, we only test whether the system is able to solve the task from a single specific start state. Generally, we would seek solutions applicable to larger regions of the state space. Our algorithm can naturally handle this in two ways: a wider *distribution* of start state can explicitly be specified, or alternatively, multiple paths with different start states (or distributions) can be simulated. Both of these approaches work successfully, although we do not report the results here due to lack of space.

5 Discussion

We have demonstrated that our model-based Bayesian RL algorithm is able to learn a policy for the cart-pole problem from scratch in a handful of iterations. The algorithm carefully models uncertainties about the underlying latent dynamics and takes them seriously into account during planning. In contrast to Doya's results reported in [14], our algorithm is very fast and can solve the swing-up task based on a minute or less interactions with the real system.

It is interesting to speculate what aspects of the model are key to its success. Although we have not yet investigated this fully, we did assess whether it is possible to solve the problem using a deterministic model for the dynamics. We repeated the experiment, but changed the predictions to have zero variance (which corresponds to a deterministic dynamics model). In this case, the algorithm failed completely to solve the task. In particular, for the deterministic model, the policy optimization problem becomes very difficult. This is presumably caused by actions having effects over long horizons, even when the dynamics models are so poor that very little can in fact be predicted. In contrast, the probabilistic model "knows" when it loses track of the state, and thus the error signal gets automatically "smoothed out". In practice, for the deterministic system the planning never finds trajectories with low expected cost, although inserting the policy from the probabilistic system yields a low cost. These local minima problems are very severe: In the cart-pole task, we have never managed to get the deterministic system to find a good solution, whereas the probabilistic system has never failed. Demos of the task with a deterministic dynamics model can also be found at http://mlg.eng.cam.ac.uk/carl/ewrl08/.

5.1 Current Limitations

Although our system works very well on the simple cart-pole problem, there are a number of ways in which the current implementation is limited:

Firstly, the system currently relies strictly on exploitation. When it has obtained a reasonable strategy it never veers from this. Thus, the strategy found is sometimes not close to an optimal strategy. For example, the experimental results reported in the previous section find a solution which swings left, then right and then left and up to balancing. In other equivalent runs, we have also seen a more direct strategy, such as left, then right up to balance. We never seem to find solutions worse than the one reported here, but there may be quite some sensitivity to the initial experience (which is random in our case). A principled solution allowing for explorations could be achieved within the current framework, by using a cost depending on both the expected cost and the variance of the cost. The variance of the cost can also be computed in closed form.

Secondly, our current approach learns very fast in terms of the amount of experience required to solve the task, but the computational demand is not negligible. In our current implementation, the optimization of the policy takes about 1 hour of CPU time. Performance can certainly be improved by writing more efficient code, and by speeding up the basic GP predictions using sparse approximations, for instance, the recent methods described in [13]. Nevertheless, it is not obvious that the scheme can necessarily learn in real time. However, once the policy has been learned, the computational requirements of applying the policy to a control task are trivial.

Thirdly, we have demonstrated learning in the special case where the observations are corrupted by only a very modest amount of observation noise. In principle, there is nothing to hinder the use of the algorithm when observations are very noisy, but one would probably have to estimate the current state more carefully—in the current setup we just assume that the state is measured exactly.

Finally, in an actual implementation, one would perhaps prefer a piecewise linear policy over the currently implemented piecewise constant policy. This could be achieved by formally treating the previously applied force as a component of the state.

6 Conclusions and Outlook

We have developed a framework based on flexible probabilistic non-parametric models for fast reinforcement learning in continuous state and action systems. The framework is conceptually simple, relying on well-established ideas, but a decisive difference is that we use fully probabilistic models of the world dynamics.

We have demonstrated the effectiveness of our approach on the cart-pole problem. Our algorithm finds a good policy from scratch using less than a minute worth of experiences—as far as we know this is an unprecedented speed for this kind of problem, which has previously been considered very hard to learn. Moreover, we require only very few assumptions: 1) we set the sampling time to be 200 ms, that is, somewhat shorter than the eigen-frequency of the system, 2) we set the trajectory length to be 5 s, that is, somewhat longer than the eigen-frequency and 3) we set the length-scale for the cost function to be $c = 0.25\,\text{m}$. We have not experimented with other settings, but believe that our system is fairly insensitive to these, as long as the order of magnitude is reasonable.

It is our belief that the success of our algorithm stems from the principled approach to handling the model's uncertainty. We anticipate that the effective solutions of more complex problems will also be possible using this algorithm. In the near future, we will explore how our algorithm performs on more challenging tasks, especially in systems with higher dimensional states. In particular, we believe that principled algorithms to address the exploration versus exploitation trade-off and other fundamental problems in practical algorithms for reinforcement learning will require careful quantification of model uncertainty.

Acknowledgements

We thank Jan Peters for many helpful discussions. MPD acknowledges support by the German Research Foundation (DFG) through grant RA 1030/1-3 to CER.

References

1. Sutton, R.S.: Integrated Architectures for Learning, Planning, and Reacting Based on Approximate Dynamic Programming. In: Proceedings of the Seventh International Conference on Machine Learning, Morgan Kaufman Publishers (1990) 215–224
2. Atkeson, C.G., Santamaría, J.C.: A Comparison of Direct and Model-Based Reinforcement Learning. In: Proceedings of the International Conference on Robotics and Automation. (1997)
3. Atkeson, C.G., Schaal, S.: Robot Learning from Demonstration. In: Proceedings of the 14th International Conference on Machine Learning, Nashville, TN, USA, Morgan Kaufmann (July 1997) 12–20
4. Abbeel, P., Quigley, M., Ng, A.Y.: Using Inaccurate Models in Reinforcement Learning. In: Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, USA (June 2006) 1–8
5. Poupart, P., Vlassis, N.: Model-based Bayesian Reinforcement Learning in Partially Observable Domains. In: Proceedings of the International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, FL, USA (January 2008)
6. Schaal, S.: Learning From Demonstration. In: Advances in Neural Information Processing Systems 9. The MIT Press, Cambridge, MA, USA (1997) 1040–1046
7. Abbeel, P., Ng, A.Y.: Exploration and Apprenticeship Learning in Reinforcement Learning. In: Proceedings of th 22nd International Conference on Machine Learning, Bonn, Germay (August 2005) 1–8
8. Peters, J., Schaal, S.: Learning to Control in Operational Space. The International Journal of Robotics Research **27**(2) (2008) 197–212
9. Kuss, M.: Gaussian Process Models for Robust Regression, Classification, and Reinforcement Learning. PhD thesis, Technische Universität Darmstadt, Germany (February 2006)
10. Rasmussen, C.E., Kuss, M.: Gaussian Processes in Reinforcement Learning. In: Advances in Neural Information Processing Systems 16. The MIT Press, Cambridge, MA, USA (June 2004) 751–759
11. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA (2006)
12. Girard, A., Rasmussen, C.E., Quiñonero Candela, J., Murray-Smith, R.: Gaussian Process Priors with Uncertain Inputs—Application to Multiple-Step Ahead Time Series Forecasting. In: Advances in Neural Information Processing Systems 15. The MIT Press, Cambridge, MA, USA (2003) 529–536
13. Snelson, E., Ghahramani, Z.: Sparse Gaussian Processes using Pseudo-inputs. In: Advances in Neural Information Processing Systems 18. The MIT Press, Cambridge, MA, USA (2006) 1257–1264
14. Doya, K.: Reinforcement Learning in Continuous Time and Space. Neural Computation **12**(1) (January 2000) 219–245