Karlsruhe Institute of Technology

Recursive Surrogate-Modeling for Stochastic Search

Bachelor's Thesis
of

Klaus Philipp Theyssen

KIT Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Autonomous Learning Robots (ALR)

Referees: Prof. Dr. Techn. Gerhard Neumann
Prof. Dr. Ing. Tamim Asfour

Advisor: M.Sc. Maximilian Hüttenrauch

Duration: November 27th, 2020 — March 26th, 2021

Erklärung

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den 26. März 2021

Klaus Philipp Theyssen

Zusammenfassung

Roboter werden in den nächsten Jahrzehnten zunehmend unseren Alltag durchdringen. Sie werden ein breites Spektrum an Aufgaben wie Altenpflege, Such- und Rettungsaufgaben und allgemeine Assistenz im täglichen Leben übernehmen können. Derzeit sind die meisten Roboter darauf angewiesen von einem erfahrenen menschlichen Bediener programmiert zu werden. Damit Roboter Teil des täglichen Lebens werden können, müssen sie sich in verändernden Umgegebungen zurechtfinden und sich an neue Situationen anpassen. Daher werden selbstlernende Roboter zunehmend an Bedeutung gewinnen. Die Forschung zu autonom lernenden Robotern ist bereits sehr aktiv und wird weiter an Bedeutung gewinnen. Es besteht eine enge Beziehung zwischen Reinforcement Learning und Robotik, wobei sich beide Forschungsgebiete gegenseitig ergänzen. Eine wichtige Herausforderung ist es, dateneffiziente Methoden für selbstlernende Roboter zu entwickeln, da Stichproben aus realen Interaktionen sehr kostspielig sind. Policy-Suchmethoden, ein Teilgebiet des Reinforcement Learning, können eingesetzt werden, um verschiedene Aufgaben durch Versuch und Irrtum zu erlernen.

In dieser Arbeit versuchen wir, die Dateneffizienz eines Policy-Suchalgorithmus zu verbessern. Dafür benutzen wir rekursive Schätzmethoden wie den Kalman Filter, der aus einer Bayes'schen Perspektive eingeführt wird. Wir implementieren verschiedene Versionen von Filteralgorithmen und vergleichen sie mit bisherigen Methoden und testen unsere Algorithmen auf Optimierungs-Testfunktionen und simulierten Robotik Aufgaben.

Abstract

Robots will increasingly permeate our daily life over the next few decades. Potentially fulfilling a wide range of tasks like elder care, search and rescue and general assistance in daily life. Today, most robots rely on being taught and programmed by a skilled human operator. For robots to become part of daily life, they need to cope with a changing environment and regularly adjust to new situations. For mastering these challenges autonomously learning robots pose a promising solutions. Currently, robot learning is already a very active field of research. There is a close relationship between reinforcement learning and robotics where both fields of research complement each other. A big challenge for robot learning is sample efficiency because real world samples are costly to obtain. Policy search methods, a sub-field of reinforcement learning, can be used to learn different tasks simply through trial and error.

In this thesis we try to improve the sample efficiency of a policy search algorithm. This is done using classical recursive estimation techniques like the Kalman filter, introduced from a Bayesian perspective. We implement different versions of filtering algorithms and compare them with previous methods and benchmark them on optimization test functions and various simulated robot tasks.

Table of Contents

Chapter 1.

Introduction

1.1. Motivation

Robots are already used extensively in industry to form production chains, where they perform the same task over and over. These robots are being programmed and fine tuned by a human engineer which requires experience and expertise.
Recently, there has been a new development of robots becoming part of our daily life's, for example in the form of vacuum cleaners and lawn mowers. In the future areas like care giving and everyday assistance and household work may become successful application domains of robotics (Schaal, 2007). This poses a dramatic shift from the prior uses of robots in industry, where they mainly worked in isolated and predefined contexts. Especially countries like Japan, facing the problem of an aging population, put increasing effort to make robots viable in these new application domains.
In daily life, the robots are confronted with different challenges, like adapting to different lighting conditions and objects being moved around. The robots which are currently on the market like vacuum cleaners and lawn mowers have either a "one size fits all" approach or need a special setup in software or hardware. To solve this issue, machine learning and especially reinforcement learning (RL) will be key technologies to enable robots to adjust to dynamic and stochastic environments.

Robotics and reinforcement learning complement each other with robotics providing a real world testing ground and reinforcement learning providing the framework for formulating problems and finding solutions. The relationship may be similar to the one of math and physics (Kober et al., 2013). Compared to other domains, in which reinforcement learning has been successfully employed, robotics poses a unique set of challenges. This makes it

necessary to explore methods which are adjusted to the inherent requirements of robotics, which include:

- a high dimensional state and action space,
- problem of obtaining real world samples,
- problem of goal specification,
- and dealing with under-modeling and model uncertainty.

Robots have the potential to transform our society. By bringing robots from the factories into our homes there will be many possibilities for improvement. To reap the benefits of this development we will have to overcome many technical and ethical challenges alike.

1.2. Contribution

Policy search algorithms have shown promise as an alternative to value function-based reinforcement learning, especially for learning motor skills in robotics (Deisenroth et al., 2013). The MORE algorithm is introduced in Abdolmaleki et al. (2015) as a stochastic search algorithm that can be used for policy search. The key idea of MORE is to learn a surrogate model of the objective function to efficiently compute the updates to the policy in closed form. One of the contributions of this thesis is to explore recursive estimation for learning the surrogate model with the goal of increasing the sample efficiency of the MORE algorithm. We focus on classical methods of parameter estimation and filtering like the Recursive Least Squares algorithm and the Kalman filter, considering more advanced methods is part of future work. We benchmark our version of the MORE algorithm on an optimization test function, a simple planar reaching task and on a simulation of a ball-in-a-cup task using the barret WAM robot arm and compare them to previous results. While we could improve the sample efficiency on the optimization test function, we did not reach state-of-the-art results for our robotic tasks. Nevertheless, we could successfully learn the robotic tasks demonstrating that the recursive surrogate-modeling approach may have the potential to eventually yield superior results with more time and research.

1.3. Structure of the Thesis

The remainder of this thesis is structured as follows:

Chapter 2: In the fundamentals chapter we lay the foundation for the thesis, first introducing the basics of reinforcement learning and then focusing on the specifics of applying RL to

robotic tasks. Then we discuss policy search as one method for solving the robot learning problem. Next we introduce the original MORE algorithm and look at Bayesian filtering. Finally we discuss some data preprocessing techniques used.

Chapter 3: Review of the related work in the field.

Chapter 4: In this chapter, we first motivate the idea of using recursive estimation for learning the surrogate model. Then we present a batch solution for the surrogate-model estimation task, which we use for comparison in our experiments. We present our approach of connecting the MORE algorithm with recursive estimation techniques for the surrogate model.

Chapter 5: In the evaluation chapter we conduct experiments with our algorithms on several test functions and some simple robot tasks. We compare our algorithm with the original MORE algorithm and the least squares approach for learning the surrogate model.

Chapter 6: We conclude the thesis with a summary of the achieved results and an outlook on future work.

Chapter 2.

Fundamentals

This chapter introduces basic concepts used throughout this thesis. First, we discuss the basics of reinforcement learning and the problem of robot learning. Next, we give an overview of policy search, a sub-field of reinforcement learning, as one method to solve the robot learning problem. The Kullback-Leibler divergence (KL) is introduced as an important information theoretic measure of similarity metric between probability distributions (Kullback and Leibler, 1951). Having discussed the underlying basics, we can introduce the MORE algorithm (Abdolmaleki et al., 2015). Next we will look at filtering from a Bayesian Estimation viewpoint and review the Kalman filter for parameter estimation. Finally, we look at some data preprocessing techniques employed.

2.1. Reinforcement Learning

Reinforcement Learning (RL) is a subfield of Machine Learning concerned with agents learning to interact with their environment and solving specific tasks. This is done through exploration and trial-and-error. The agents try to discover cause and effect relationships between their actions. Compared to supervised learning and unsupervised learning it more closely resembles the way we humans learn by interacting with our environment.
As Sutton and Barto (2018) puts it, the term reinforcement learning relates to a class of problems, solution methods and the field that studies these problems and solutions. Some success stories of RL include mastering the game of Go (Silver et al., 2016) and Atari games (Mnih et al., 2013). Generally, RL is applicable to a large range of problems. In supervised learning the best action is presented to the system, whereas the agent in a reinforcement learning setting only receives a reward (or punishment) for each action. To gain information

about the rewards the agent needs to explore previously unused actions. The decision of daring to try new things or to keep performing safe well-known actions is known as the *exploration-exploitation tradeoff*. In general, the agent should exploit known actions that give decent reward, but he first has to try different things to learn about these actions, and then he has to progressively focus in on them. The reward signal is typically a single scalar value, hence the amount of information the agent receives is minimal compared to supervised and unsupervised learning approaches.

The classical approach to formalizing problems in RL is through Markov Decision Processes (MDPs). MDPs are a mathematical framework for decision making in deterministic and stochastic environments. MDPs focus on only three aspects - sensation, action and goal, which are central for reinforcement learning problems. MDPs satisfy the Markov property, which states that "the future is independent of the past given the present". In our case, this means the next state s' and the reward only depend on the previous state s and action a (Sutton et al., 1992). An MDP can be formally defined as a tuple (S, A, P, r):

- a set of states $s \in S$ that describe the environment,
- a set of actions $a \in A$ that can be performed by the agent in the environment
- a transition function $P(s_{t+1} | s_t, a_t)$ that gives the probability of a new state s_{t+1} after an action a_t has been taken in state s_t,
- and a reward function $r(s_t, a_t)$ that specifies the immediate reward after taking action a_t in state s_t.

The Markov property can be expressed as

$$P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1} \dots) = P(s_{t+1} | s_t, a_t).$$

This recapitulates the notion of state - a state is a sufficient statistic for predicting the future, rendering previous observations irrelevant. In robotics, we may only find some approximate notion of state.

We model the agent and its environment as a state $s \in S$. The agent may perform action $a \in A$ which can be either discrete or continuous. For every action step, the agent receives a reward R, which is a scalar value. The overarching goal is to find a mapping from states to actions, called policy π, that picks actions in a way that the reward is maximized. We can distinguish an *episodic* setting from an on-going task. In the episodic setting the task is restarted after the end of an episode and the goal is to maximize the total reward per episode. In on-going tasks the goal is to simply achieve high average reward over the whole life-time or one can use a formulation with a discounted return (weighting the future and past differently).

Generally, the goal is to find an optimal policy π^*, a mapping from states to actions that maximizes the expected return J. Optimal behavior can be modeled in different ways, resulting in different definitions for expected return:

- for a finite-horizon model with horizon H we get

$$J = E\left\{\sum_{t=0}^{H} R_t\right\},$$

- a discounted reward can be modeled with a discount factor γ (with $0 \leq \gamma < 1$), which yields

$$J = E\left\{\sum_{t=0}^{\infty} \gamma^t R_t\right\}.$$

Reinforcement learning can also be seen as general case of optimal control as in Sutton et al. (1992). While optimal control assumes perfect knowledge, RL uses approximations and data-driven techniques.

2.2. Robot Learning

The sub-field where reinforcement learning and machine learning intersect with robotics is called *Robot Learning*. It aims to bridge the gap between programmed robots, with fine tuned controllers and fully autonomous robots. As proposed in Deisenroth et al. (2013), robot control can be modeled as a reinforcement learning problem.
The state space $\mathbf{x}$ in robotic tasks is high dimensional and made up of the internal state of the robot (e.g., joint position, body position, camera images) and external state (e.g. object locations, lighting). The true state is not observable and also not noise free. The robot chooses its next motor control u according to a policy π. This policy π may be deterministic $u = \pi(x)$ or stochastic $u \sim \pi(x,u) = P(u|x)$. The motor command u alters the state according to the probabilistic transition function $p(\mathbf{x}_{t+1}|x_t, u_t)$. This transition function is not known, in model-based policy search this function is learned from data and used to improve the policy. Collectively, the states and actions of the robot form a trajectory $\tau = (x_0, u_0, x_1, u_1, ...)$ which is also called a rollout or a path. There has to be a numeric scoring system assessing the quality of the robots trajectory and returning a reward signal $R(\tau)$. For episodic learning tasks, the task ends after a given number T time steps. Then, the accumulated reward $R(\tau)$ for a trajectory is given by

$$R(\tau) = r_T(x_T) + \sum_{t=0}^{T-1} r_t(x_t, u_t)$$

where r_t is an instantaneous reward function (e.g. a punishment for energy consumed) and r_T the final reward. When performing a reaching task this may take the form of a quadratic punishment term for deviation from the goal posture.

If we consider an infinite-horizon for an on-going task we get

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(x_t, u_t)$$

where $\gamma \in [0, 1)$ is a discount factor that discounts rewards further in the future.

Many tasks in robotics can thus be formulated as choosing an optimal control policy π^* that maximizes the expected accumulated reward

$$J_\pi = \mathbb{E}[R(\tau)|\,\pi] = \int R(\tau) p_\pi(\tau) d\tau$$

where $R(\tau)$ defines the objectives of the task, and $p_\pi(\tau)$ the distribution over trajectories τ.

Formulating robotic tasks in this way allows us to apply methods from reinforcement learning to them.

2.2.1. Challenges

Except for simple tabular problems reinforcement learning can generally be considered a difficult problem. One reason for this is that the reward signal may be given only occasionally and even then it may be unclear which of the agents actions were responsible for a certain reward signal.

Robotics is different compared to other fields where reinforcement learning is used. First of all, the states and actions of the robots in the real world are inherently continuous requiring us to deal with the resolution into a discrete representation. In addition, the state space can have a high dimensionality and working with real world systems on real hardware is costly and makes manual interventions necessary. Robots require algorithms to run in real-time and working with real sensors further introduces discrepancy between sensing and execution. Generally, the state represented by sensors slightly lags behind the real state due to processing and communication delays. This is in stark contrast to most other reinforcement learning algorithms, which assume actions to take effect instantaneously.

Most traditional methods from RL like TD-learning (Sutton and Barto, 2018) have been unfit for these particular requirements of robotic tasks. Stressing the importance of robotics as a special testing ground for RL that demands new developments and innovative research. We will now discuss some problems encountered when applying RL to robotics. This treatment focuses only on a certain set of challenges and is not meant to be exhaustive.

Curse of Dimensionality

The term "Curse of Dimensionality" was coined by Bellman (1957) when he explored optimal control in higher dimensions and encountered an exponential explosion of the states and actions. For example, in our evaluation we run a simulation of a simple planar reaching task with a 5 link robot arm using Dynamic Movement Primitives (Ijspeert et al., 2002) for policy representation and get a 25 dimensional state. Especially modern anthropomorphic robots tend to have many degrees of freedom, which further increases the dimensionality.

The agent in RL generally needs to collect data throughout the entire state-space to ensure global optimization. In robotics the agent often has to deal with high dimensional states and actions due to many degrees of freedom which makes this global optimization in many cases infeasible. To alleviate this problem a widespread idea is to use expert demonstration to get a good initialization for the agent's policy. This eliminates the need to explore the entire search space, instead the agent can focus on locally optimizing the initial policy. This concept of imitation learning focuses on the problem of "learning from demonstrations" which plays an important role for robotics (Osa et al., 2018). Imitation learning will enable domain experts to teach motions and skills without special knowledge about robotics, which will be crucial when robots start making their way from factories into everyday life.

Curse of Real-world Samples

Robot hardware is expensive and suffers from wear and tear, making costly maintenance necessary. Hence, safe methods for real robots should avoid big jumps in policy updates, as such sudden changes may result in unpredictable movements and consequently damage the robot. This constraint is commonly referred to as *safe exploration*. Whereas in traditional reinforcement learning safe exploration does not receive much attention, it has become a key issue for robot learning (Schneider, 1997).

When working with a real physical robot, different external factors like temperature may change the robot's dynamics and additionally uncertainty from the sensors makes it difficult to reproduce and compare results.

Most tasks also require a "human in the loop" who either supervises the robot or resets the setup after one episode. Even if this step can be automated the data generation is very slow compared to other applications of RL which are based only on simulation. For a single robot the training time has a natural limit and in total only relatively few executions can be completed. One method for gathering more data is "collective robot learning" described in Kehoe et al. (2015). The idea is for multiple robots to share their data on trajectories, policies and outcomes. Currently, this seems only viable for large corporations with significant capital and therefore stresses the importance of developing sample efficient algorithms. Thus, in robot learning the constraint of using only a small number of trials is given more weight than limiting memory consumption or computational complexity.

Curse of Goal Specification

Defining a good reward function in robot reinforcement learning is difficult and often needs a lot of domain knowledge and expertise. There are certain trade-offs to keep in mind, for example performing a powerful swing for a hitting task may yield high reward but may damage or shorten the life-time of the robot. Further, reinforcement learning algorithms may solve tasks in unintended ways by exploiting the reward function in an unforeseen fashion (Ng et al., 1999). An alternative to specifying the reward function manually is Inverse Reinforcement Learning (Russell, 1998) The goal of inverse reinforcement learning is to recover the unknown reward function from the expert's trajectories.

The discussed challenges illustrate the difficulty of the optimal control problem in robotics. Generally, autonomous learning algorithms are rarely employed on robots for real daily usage. Also, most algorithms are over fitted to a particular robot architecture and do not generalize to other robots easily. Even minor flaws or errors in the employed method can completely prevent success of learning. Nonetheless, appropriately chosen algorithms and rewards functions can already achieve promising results (Kober et al., 2013), but as a large number of different methods exist there is no clear general recipe for robot learning, underlining the fact that robot learning still has a lot of open problems, and will continue to grow as a research field.

2.2.2. Policy Search

Many traditional methods in RL try to estimate the expected long-term reward of a policy for each state $\mathbf{x}$ and time step t, which leads to formulation of the value function $V_t^{\pi}(\mathbf{x})$. With the value function we can asses the quality of executing action $\mathbf{u}$ in state $\mathbf{x}$. This assessment is used to directly compute the policy by action selection or to update the policy π. As value function methods typically require to fill the complete action-space they struggle with the high dimensionality encountered in robotics. Thus, policy search methods have become more common for applying RL to robotics. Policy search methods opposed to value-based methods use parameterized policies π_θ to search directly in the parameter space Θ of the policies (with $\theta \in \Theta$). This allows using RL with high-dimensional continuous action spaces encountered in robotics by reducing the search space of possible policies. Policy search further allows the usage of predefined task-appropriate policy representations like Dynamic Movement Primitives (Schaal et al., 2005), as well as easily integrating imitation learning for policy initialization.

Generally, we can divide policy search into model-free and model-based and differentiate whether stochastic or deterministic trajectories are used. Model-free policy search uses trajectories from the robot directly for updating the policy. Model-based methods use the data from the robot to first learn a model of the robot. This model is then used to generate trajectories that are used for policy updates. Due to their simplicity and by avoiding the need

of learning a model, which further introduces the problem of under-modeling, model-free methods have been employed to a wide range of problems (Deisenroth et al., 2013).

The most important concept in policy search is computing the policy updates. Both model-free and model-based policy search methods use policy gradients (PG) which employ gradient ascent for maximizing the expected return. The REINFORCE algorithm introduced in Williams (1992) is an example for a model-free method using policy gradients. The PILCO (probabilistic inference for learning control) policy search framework (Deisenroth and Rasmussen, 2011) is an example for a model-based approach. Another way to compute the policy updates is using the Expectation-Maximization algorithm (Bishop, 2006), by formulating policy search as an inference problem. One model-free method for this category is the Policy learning by Weighting Exploration with the Returns (PoWER) algorithm (Kober and Peters, 2011). The third category of policy updates we look at are information-theoretic (Inf. Th.) approaches based on the Kullback-Leibler Divergence (Section 2.3). The relative entropy policy search algorithm (REPS) (Peters et al., 2010) and the MORE algorithm (Section 2.4) fit into this category. In Figure 2.1 we give a graphical overview of the different policy search methods.

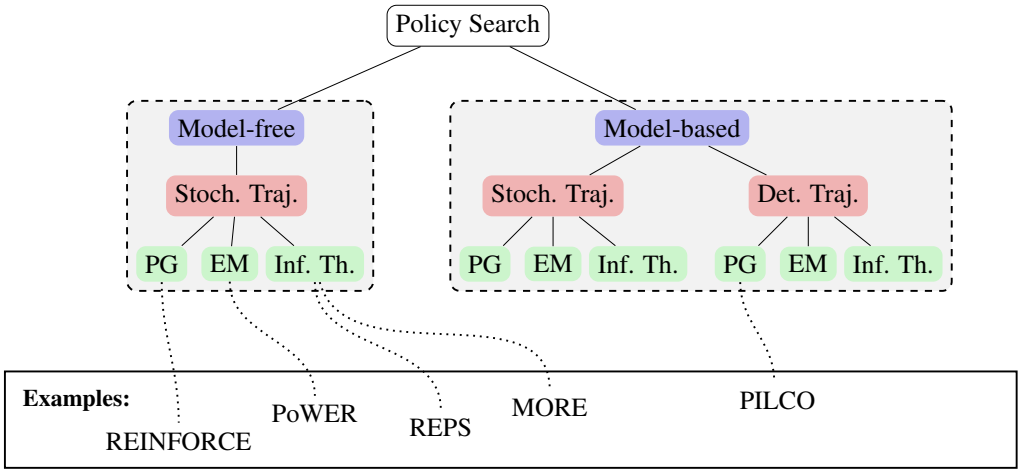


Figure 2.1.: Categorization of policy search into model-free and model-based with differentiation based on trajectory generation and policy update methods. For model-free methods stochastic trajectory (Stoch. Traj.) generation directly samples the trajectories from the robot and in the model-based case from the learned model of the robot. Some model-based methods also use deterministic trajectory (Det. Traj.) prediction where the goal is to analytically predict the trajectory distribution instead of sampling from the system. The drawing is based on Deisenroth et al. (2013).

In this thesis, we will focus on model free policy search methods where the trajectories are generated by "sampling" from the robot and information theoretic policy updates are used. More specifically we will formulate our approach in the setting of stochastic search algorithms, which are general black-box optimizers. They are applied in a wide range of

fields like operations research and machine learning. Since these algorithms do not use any knowledge about the objective function it is straightforward to apply them to policy search in the episode-based formulation.

Using stochastic search algorithms we keep an upper-level policy $\pi_\omega(\theta)$ which selects the parameters of the actual control policy $\pi_\theta(\mathbf{u}|\mathbf{x})$ of the robot. Instead of directly finding the parameters θ of the lower-level policy we want to find the parameter vector ω which defines a search distribution over θ. We can then use this search distribution to directly explore the parameter space.

2.3. Kullback-Leibler (KL) Divergence

Several algorithms that can be used for model-free policy search like NES (Wierstra et al., 2014) and REPS (Peters et al., 2010) rely on the Kullback-Leibler divergence, also known as the relative entropy, for controlling the difference between the old and updated policy. Working with real robots additionally requires to perform safe exploration. Big exploration steps may result in damaging the hardware. Specifically, it measures the Shannon entropy of one distribution relative to the other. The KL divergence from q to p is defined as

$$\mathrm{KL}(p||q) = \int p(\theta) \log\left(\frac{p(\theta)}{q(\theta)}\right) d\theta$$

where p and q are continuous probability distributions. Note, that in general the relative entropy is not symmetric under interchange of the distributions p and q.

2.4. MORE Algorithm

Model-Based Relative Entropy Stochastic Search (MORE) (Abdolmaleki et al., 2015) is a stochastic search algorithm that can be used as a policy search method for episodic reinforcement learning tasks. The key idea is using information-theoretic policy updates by bounding the relative entropy (Kullback Leibler divergence) between two subsequent policies. As MORE uses no gradient information and requires only function evaluations of the objective function it can be seen as a type of stochastic search algorithm and can be used for black box optimization problems. The essential difference of MORE compared to previous algorithms using the KL-bound like REPS (Peters et al., 2010) lies in utilizing a quadratic surrogate model of the objective function to satisfy the KL-bound in closed form without approximations. On top of that it introduces a lower bound constraint on the entropy of the new distribution to avoid premature convergence.

The MORE algorithm can be used to maximize an objective function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$. The goal is to find one or more parameter vectors $\mathbf{x} \in \mathbb{R}^n$ with the highest possible objective

value. For this the algorithm maintains a search distribution over $\mathbf{x}$ for which the expectation over the reward is maximized. To accomplish this, the MORE algorithm iteratively draws samples from the search distribution, implemented as a multivariate Gaussian distribution, i.e. $\pi(\mathbf{x}) = \mathbf{N}(\mathbf{x}|\,\mu,\Sigma)$. This distribution corresponds to an upper-level policy and is parameterized by the mean and covariance matrix. In each iteration, N samples are drawn from the search distribution. Each sample $\mathbf{x}$ is then evaluated on the objective function yielding the corresponding objective value r, this constitutes the data $\{\mathbf{x}^k, r^k\}_{k=1\ldots N}$ which is used to compute a new search distribution. This iterative process is run until the algorithm converges.

2.4.1. MORE Framework

To satisfy the bound on the relative entropy between subsequent search distributions and the bound on the entropy while optimizing the objective function we use the theory of constraint optimization (Boyd et al., 2004). In constraint optimization we want to maximize a function $f(x)$ under a set of equality constraints and inequality constraints.

$$\begin{aligned} \text{maximize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) = 0, \quad i = 1,\ldots,m \\ & c_j(x) \leq 0, \quad j = 1,\ldots,p \end{aligned} \tag{2.1}$$

With this we can now formulate the constraint optimization problem to obtain a new search distribution that maximizes the expected objective value while upper-bounding the KL-divergence and lower-bounding the entropy of the search distribution at the same time

$$\begin{aligned} \max_{\pi} \quad & \int \pi(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}, \\ \text{subject to} \quad & \text{KL}\,(\pi(\mathbf{x})||q(\mathbf{x})) \leq \varepsilon, \\ & H(\pi) \geq \beta, \\ & 1 = \int \pi(\mathbf{x}) d\mathbf{x}, \end{aligned}$$

where $H(\pi) = -\int \pi(\mathbf{x}) \log \pi(\mathbf{x}) d\mathbf{x}$ denotes the entropy of the search distribution. The parameters ε and β are hyperparameters to control the exploration-exploitation trade-off. The ε can be chosen freely to control the step size of the policy update. Generally, its size will depend on the specific problem and the amount of available samples. The bound β is defined such that the relative difference between the entropy of the policy $H(\pi)$ and a minimum exploration policy $H(\pi_0)$ is decreased for a certain percentage:

$$H(\pi) - H(\pi_0) \geq \gamma(H(q) - H(\pi_0)) \rightarrow \beta = \gamma(H(q) - H(\pi_0) + H(\pi_0)$$

The key idea of the MORE algorithm is to use a surrogate model of the objective function for satisfying the bound on the KL-divergence. The surrogate model has the following form

$$f(\mathbf{x}) \approx \hat{f}(\mathbf{x}) = \mathbf{x}^T \mathbf{R}\mathbf{x} + \mathbf{x}^T \mathbf{r} + r.$$

A quadratic model is sufficient, since the exponent of a Gaussian distribution is also quadratic and thus it would not be possible to exploit the additional information of a more complex surrogate model. For estimating the surrogate model the original approach of MORE employs a Bayesian dimensionality reduction method and linear regression.

By using the theory of Lagrange multipliers and duality (Boyd et al., 2004) we obtain the solution to the constraint optimization problem (for details see Appendix A.1). Assuming π to be Gaussian, solving the dual problems yields

$$\begin{aligned} \pi_{t+1} &= \mathrm{N}(\mu_{t+1}, \Sigma_{t+1}) \\ \mu_{t+1} &= (\eta\Sigma_t^{-1}\mu_t + \mathbf{r})/(\eta+\omega) \\ \Sigma_{t+1} &= (\eta\Sigma_t^{-1} + \mathbf{R})/(\eta+\omega), \end{aligned} \tag{2.2}$$

where η and ω are the Lagrangian multipliers obtained by minimizing the corresponding dual function. In addition, the linear term $\mathbf{r} \in \mathbb{R}^n$ and the quadratic term $\mathbf{R} \in \mathbb{R}^{n\times n}$ of the current surrogate model are used to update the policy. With these equations, we can iteratively update the search distribution.

2.5. Bayesian Filtering

Optimal filtering is concerned with estimating the state of a time-varying system which is indirectly observed through noisy measurements. This section will focus on optimal filtering from a Bayesian perspective and is largely based on Särkkä (2013).

The term "Bayesian" refers to inference methods that represent "degrees of certainty" using probability theory. They are fundamentally based on applying Bayes' rule to update the degree of certainty given data. More generally, as Gelman et al. (2013) puts its, Bayesian inference is the process of fitting a probability model to a set of data and summarizing the result by a probability distribution on the parameters of the model and on unobserved quantities such as predictions for new observations.

Filtering methods are widely used in robotics to deal with noisy sensor measurements. This includes tasks like object tracking, robot control and robot localization (Chen, 2011). Since robots need to make decisions based on relatively small amounts of data, it is common to adopt a Bayesian perspective when using filtering methods and for reasoning about the environment in general (Thrun, 2002).

2.5.1. Bayesian Parameter Estimation

In general, when using Bayesian models for estimating *unknown parameters* θ, the following probability distributions are used:

- **Prior Distribution:** Encodes the information on parameter θ before seeing any observations. When we are uncertain about our prior information we can choose a high variance of the distribution or use a non-informative prior (which imposes the minimal amount of structure on the data).

$$p(\theta) = \text{information on parameter } \theta \text{ before seeing any observations}$$

- **Measurement Model:** Models the relationship between true parameters and the measurements.

$$p(y|\,\theta) = \text{distribution of observation } y \text{ given the parameters } \theta$$

- **Posterior Distribution:** The conditional distribution of the parameters given the observations. It represents the updated belief about the parameters after obtaining the measurements. It can be computed by using Bayes' rule.

$$p(\theta|y) = \frac{p(y|\,\theta)p(\theta)}{p(y)} \propto p(y|\,\theta)p(\theta)$$

2.5.2. Optimal Filtering as Bayesian Inference

The goal of optimal filtering can be seen as solving a statistical inversion problem where the unknown quantity is a potentially vector valued time series $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ...\}$ which is observed through a set of noisy measurements $\{\mathbf{y}_1, \mathbf{y}_2, ...\}$, as depicted in Figure 2.2.

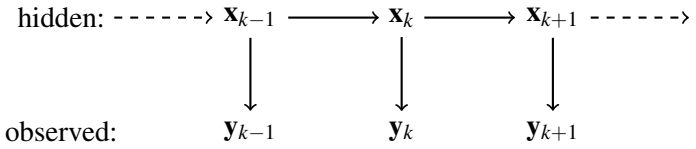


Figure 2.2.: A sequence of hidden states $\mathbf{x}_k$ is indirectly observed through noisy measurements. Drawing recreated from (Särkkä, 2013).

We want to estimate the hidden states from the observed measurements. Solving this problem in a Bayesian sense means we need to compute the joint posterior distribution of all states given all the measurements, for this we can use Bayes' rule. This yields a batch solution to

the statistical estimation problem resulting in the equation

$$p(\mathbf{x}_{0:T}|y_{1:T}) = \frac{p(y_{1:T}|\mathbf{x}_{0:T})p(\mathbf{x}_{0:T})}{p(y_{1:T})} \tag{2.3}$$

where $p(\mathbf{x}_{0:T})$ is the prior distribution defined by the dynamic model, $p(y_{1:T}|\mathbf{x}_{0:T})$ is the likelihood model for the measurements and $p(y_{1:T})$ is the normalization constant defined as

$$p(y_{1:T}) = \int p(y_{1:T}|\mathbf{x}_{0:T})p(\mathbf{x}_{0:T})d\mathbf{x}_{0:T}.$$

This formulation is unfit for dynamic estimation tasks where we receive measurements one at a time, since at each time step we would have to recompute the full posterior distribution. As the number of time steps increases, also the dimensionality of the full posterior would increase making computations intractable. If we instead only compute selected marginal distributions we can make computation feasible again. For this we need to restrict our dynamic models to probabilistic Markov sequences, with a transition probability distribution $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ that depends only on the previous state.

In *Bayesian filtering and smoothing* the following marginal distributions are of interest

- **Filtering distributions** computed by the Bayesian filter are the marginal distributions of the current state $\mathbf{x}_k$ given the current and previous measurements $y_{1:k} = \{y_1, ..., y_k\}$

$$p(\mathbf{x}_k|y_{1:k}), \quad k = 1, ..., T.$$

- **Prediction distributions** which can be computed with the prediction step of the Bayesian filter are the marginal distributions of the future state $\mathbf{x}_{k+n}$, with n steps after the current time step

$$p(\mathbf{x}_{k+n}|y_{1:k}), \quad k = 1, ..., T \quad n = 1, 2, ...$$

- **Smoothing distributions** computed by the Bayesian smoother are the marginal distributions of the state $\mathbf{x}_k$ given a certain interval $y_{1:T} = \{y_1, ..., y_T\}$ of measurements with $T > k$

$$p(\mathbf{x}_k|y_{1:T}), \quad k = 1, ..., T$$

We will focus on the filtering distribution, as we use it for our surrogate model estimation problem.

For the filtering distribution we can formulate the recursive Bayesian solution to the statistical inversion problem as follows:

1. The distribution of measurements is modeled by the likelihood function $p(y_k|\mathbf{x}_k)$ and the measurements are assumed to be conditionally independent

2. In the beginning at time step zero all information about $\mathbf{x}$ is contained in the prior distribution $p(\mathbf{x}_0)$

3. The measurements are assumed to be obtained one at a time, first y_1 then y_2 and so on. The key idea is to use the posterior distribution from the previous time step as the current prior distribution

$$\begin{aligned}
p(\mathbf{x}_1|y_1) &= \frac{1}{Z_1} p(y_1|\mathbf{x}_1) p(\mathbf{x}_0), \\
p(\mathbf{x}_2|y_{1:2}) &= \frac{1}{Z_2} p(y_2|\mathbf{x}_2) p(\mathbf{x}_1|y_1), \\
&\vdots \\
p(\mathbf{x}_k|y_{1:k}) &= \frac{1}{Z_k} p(y_k|\mathbf{x}_k) p(\mathbf{x}_k|y_{1:k-1}), \\
&\vdots \\
p(\mathbf{x}_T|y_{1:T}) &= \frac{1}{Z_T} p(y_T|\mathbf{x}_T) p(\mathbf{x}_{T-1}|y_{1:T-1}),
\end{aligned}$$

   with the normalization term $Z_k = \int p(y_k|\mathbf{x}_k) p(\mathbf{x}_k|y_{1:k-1}) d\mathbf{x}_k$.

This recursive formulation of Bayesian estimation has several useful properties. First of all it can be considered as the *online learning* solution to the Bayesian learning problem. As each step in the recursive estimation is a full Bayesian update step, batch Bayesian inference is a special case of recursive Bayesian inference. Due to the sequential nature of the estimation we can model what happens to the unknown quantity $\mathbf{x}$. This turns out to be the basis of filtering theory, where time behavior is modeled by assuming the quantity to be a time-dependent stochastic process $\mathbf{x}(t)$.

2.5.3. Least Squares

To ease the application of Bayesian filtering to our specific problem at a later stage we will now discuss a simple regression problem. First we are going to derive the least squares solution, as we use a least squares (LS) approach for comparison in our experiments.

We consider a simple linear regression problem,

$$y_k = x_1 + x_2 t_k + \varepsilon_k \tag{2.4}$$

where we assume that the measurement noise is zero mean Gaussian with a given variance $\varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ and that the prior distribution of the parameters $\mathbf{x} = (x_1 \; x_2)^T$ is Gaussian with known mean and covariance $\mathbf{x} \sim \mathcal{N}(\mathbf{m}_0, \mathbf{P}_0)$. Note that in this section the parameters $\mathbf{x}$ are assumed to stay constant. We now want to estimate the parameters $\mathbf{x}$ from a set of measurement data $\mathcal{D} = \{(t_1, y_1), ..., (t_T, y_T)\}$.

In compact probabilistic notation the linear regression model can be written as

$$\begin{aligned} p(y_k|\mathbf{x}) &= \mathcal{N}(y_k|\mathbf{H}_k\mathbf{x},\sigma^2) \\ p(\mathbf{x}) &= \mathcal{N}(\mathbf{x}|\mathbf{m}_0,\mathbf{P}_0) \end{aligned} \tag{2.5}$$

Here $\mathbf{H}_k = (1\ t_k)$ is the design matrix and contains the regressors and $\mathcal{N}(\cdot)$ denotes the Gaussian probability density function (see Appendix A.2). The row vector $\mathbf{H}_k$ is denoted in matrix notation, to avoid using different notation for scalar and vector measurements. The batch solution can then be easily obtained by application of Bayes'rule

$$\begin{aligned} p(\mathbf{x}|y_{1:T}) &\propto p(\mathbf{x})\prod_{k=1}^{T} p(y_k|\mathbf{x}) \\ &= \mathcal{N}(\mathbf{x}|\mathbf{m}_0,\mathbf{P}_0)\prod_{k=1}^{T}\mathcal{N}(y_k|\mathbf{H}_k\mathbf{x},\sigma^2). \end{aligned}$$

Because the prior and likelihood are Gaussian, the posterior distribution in turn is also Gaussian and denoted by

$$p(\mathbf{x}|y_{1:T}) = \mathcal{N}(\mathbf{x}|\mathbf{m}_t,\mathbf{P}_t).$$

By completing the quadratic form in the exponent we get the equations (2.6) for the mean and covariance of the posterior distribution.

$$\begin{aligned} \mathbf{m}_T &= \left[\mathbf{P}_0^{-1} + \frac{1}{\sigma^2}\mathbf{H}^T\mathbf{H}\right]^{-1}\left[\frac{1}{\sigma^2}\mathbf{H}^T\mathbf{y} + \mathbf{P}_0^{-1}\mathbf{m}_0\right] \\ \mathbf{P}_T &= \left[\mathbf{P}_0^{-1} + \frac{1}{\sigma^2}\mathbf{H}^T\mathbf{H}\right]^{-1} \end{aligned} \tag{2.6}$$

where $\mathbf{H}_k = (1\ t_k)$ and

$$\mathbf{H} = \begin{pmatrix}\mathbf{H}_1\\ \vdots \\ \mathbf{H}_T\end{pmatrix} = \begin{pmatrix}1 & t_1\\ \vdots & \vdots \\ 1 & t_t\end{pmatrix}, \qquad y = \begin{pmatrix}y_1\\ \vdots \\ y_T\end{pmatrix}$$

2.5.4. Kalman Filter

For the least squares solution, the parameters $\mathbf{x} = (x_1\ x_2)$ of the regression model (2.4) are assumed to stay constant. Now, we assume the parameters are allowed to perform a Gaussian

random walk between measurements modeled by

$$\begin{aligned} p(y_k|\,\mathbf{x}_k) &= \mathcal{N}(y_k|\,\mathbf{H}_k\mathbf{x}_k, \sigma^2) \\ p(\mathbf{x}_k|\,\mathbf{x}_{k-1}) &= \mathcal{N}(\mathbf{x}_k|\,\mathbf{x}_{k-1}, \mathbf{Q}) \\ p(\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_0|\,\mathbf{m}_0, \mathbf{P}_0) \end{aligned}$$

where $\mathbf{Q}$ is the covariance of the random walk.

We will now formulate the linear regression model as a time-invariant model by avoiding explicit covariates t_k. This has the advantage that the model is not dependent on the absolute time, but only on the relative positions of states and measurements in time. We denote the time difference between consecutive times as $\Delta t_{k-1} = t_k - t_{k-1}$. The idea is that if the underlying phenomenon (signal, state, parameter) x_k was exactly linear, the difference between adjacent time points could be written exactly as

$$x_k - x_{k-1} = \dot{x}\,\Delta t_{k-1}$$

where $\dot{x}$ is the derivative, which is constant in the linear case. As this may not exactly be the case we also add some noise to the model to get

$$\begin{aligned} x_{1,k} &= x_{1,k-1} + \Delta t_{k-1} x_{2,k-1} + q_{1,k-1} \\ x_{2,k} &= x_{2,k-1} + q_{2,k-1} \\ y_k &= x_{1,k} + s_k \end{aligned}$$

where the signal is the first component of the state ($x_{1,k} = x_k$) and the derivative is the second ($x_{2,k} = \dot{x}_k$). The noises are $s_k \sim \mathcal{N}(0, \sigma^2)$ and $(q_{1,k-1}, q_{2,k-1}) \sim \mathcal{N}(0, \mathbf{Q})$. Now, the linear regression model (2.5) can be written in the form

$$\begin{aligned} p(y_k|\,\mathbf{x}_k) &= \mathcal{N}(y_k|\,\mathbf{H}\,\mathbf{x}_k, \sigma^2) \\ p(\mathbf{x}_k|\,\mathbf{x}_{k-1}) &= \mathcal{N}(\mathbf{x}_k|\,\mathbf{A}_{k-1}\mathbf{x}_{k-1}, \mathbf{Q}) \end{aligned}$$

where

$$\mathbf{A}_{k-1} = \begin{pmatrix} 1 & \Delta t_{k-1} \\ 0 & 1 \end{pmatrix}, \qquad \mathbf{H} = (1 \quad 0).$$

In this formulation the model is a special case of generic linear Gaussian models of the form,

$$\begin{aligned} p(y_k|\,\mathbf{x}_k) &= \mathcal{N}(y_k|\,\mathbf{H}_k\mathbf{x}_k, \mathbf{R}_k) \\ p(\mathbf{x}_k|\,\mathbf{x}_{k-1}) &= \mathcal{N}(\mathbf{x}_k|\,\mathbf{A}_{k-1}\mathbf{x}_{k-1}, \mathbf{Q}_{k-1}) \end{aligned}$$

for which the Kalman Filter (Kalman, 1960) is the optimal recursive solution.

The Kalman filter equations can be expressed as prediction and update steps as follows:

- The prediction step

$$\begin{aligned} \mathbf{m}_k^- &= \mathbf{A}_{k-1}\mathbf{m}_{k-1} \\ \mathbf{P}_k^- &= \mathbf{A}_{k-1}\mathbf{P}_{k-1}\mathbf{A}_{k-1}^T + \mathbf{Q}_{k-1} \end{aligned} \tag{2.7}$$

- The update step

$$\begin{aligned} v_k &= y_k - \mathbf{H}_k\mathbf{m}_k^- \\ \mathbf{S}_k &= \mathbf{H}_k\mathbf{P}_k^T\mathbf{H}_k^T + \mathbf{R}_k \\ \mathbf{K}_k &= \mathbf{P}_k^-\mathbf{H}_k^T\mathbf{S}_k^{-1} \\ \mathbf{m}_k &= \mathbf{m}_k^- + \mathbf{K}_k v_k \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T \end{aligned} \tag{2.8}$$

2.6. Data Preprocessing Techniques

We now review several data preprocessing techniques. Some key challenges that arise in our data are high range of objective values, and dealing with sharp jumps in the reward signal from penalties (e.g collisions).

2.6.1. Whitening

Whitening is a common data preprocessing method in statistical analysis to transform a correlated random vector into an uncorrelated one (Kessy et al., 2018). We employ whitening to our algorithm, to make the optimization numerically more stable.

Whitening is a linear transformation that converts a d-dimensional random vector $\mathbf{x} = (x_1, ..., x_d)^T$ with mean $\mathrm{E}(\mathbf{x}) = \mu = (\mu_1, ..., \mu_d)^T$ and positive definite $d \times d$ covariance matrix $\mathrm{var}(\mathbf{x}) = \Sigma$ into a new random vector

$$\mathbf{z} = (z_1, ..., z_d)^T = \mathbf{W}\mathbf{x} \tag{2.9}$$

of the same dimension d and with unit diagonal "white" covariance $\mathrm{var}(\mathbf{z}) = \mathbf{I}$. The square $d \times d$ matrix $\mathbf{W}$ is called the whitening matrix.

The whitening transformation defined in Equation (2.9) requires the choice of a suitable whitening matrix W. Since $\mathrm{var}(z) = \mathbf{I}$ it follows that $\mathbf{W}\Sigma\mathbf{W}^T = \mathbf{I}$ and thus $\mathbf{W}(\Sigma\mathbf{W}^T\mathbf{W}) = \mathbf{W}$, which is fulfilled if $\mathbf{W}$ satisfies the condition

$$\mathbf{W}^T\mathbf{W} = \Sigma^{-1}$$

This constrain does not uniquely determine the whitening matrix $\mathbf{W}$, instead given Σ there are infinitely many possible matrices $\mathbf{W}$, because it allows for rotational freedom.

A widely used procedure is *Cholesky whitening* which is based on Cholesky factorization of the precision matrix $\mathbf{L}\mathbf{L}^T = \Sigma^{-1}$. This leads to the whitening matrix $\mathbf{W}^{\text{Chol}} = \mathbf{L}^T$.

2.6.2. Normalization

In our experiments we deal with data from unsmooth reward functions caused by penalties. As data with mean zero and variance makes our estimation task easier we normalize the data by computing the standard score also called z-score (Mendenhall and Sincich, 2016) for each sample x as follows

$$z = \frac{x - \mu}{\sigma}.$$

The true population mean and standard derivation are unknown so we use the sample mean and standard deviation instead.

Exponential Moving Average

For our recursive estimation approach we would like to compute the mean and variance in an incremental on-line fashion. In Finch (2009) a way to compute the exponential moving average recursively is introduced. When using an exponential moving average we apply different weighting factors to the samples that are processed. This can be controlled with the coefficient $0 < \alpha < 1$, where a higher α value discounts older samples faster. The equations for updating the mean and the variance with the sample x_i are

$$\begin{aligned} \delta_i &= x_i - \mu_{i-1} \\ \mu_i &= \mu_{i-1} + \alpha\, \delta_i \\ \sigma_i^2 &= (1-\alpha)(\sigma_{i-1}^2 + \alpha\, \delta_i^2). \end{aligned} \tag{2.10}$$

Chapter 3.

Related Work

The MORE algorithm can be used as a model-free policy search algorithm with an information theoretic approach for updating the policy. Information-theoretic policy algorithms use the Kullback-Leibler Divergence in a constrained optimization problem to bound the distance of the old policy to the new one. Whereas MORE uses a surrogate model to compute the new search distribution in closed form other methods like the relative entropy policy search algorithm (REPS) proposed in Peters et al. (2010) use a sample-based approximation for the KL-divergence. While in Peters et al. (2010) a step-based policy search algorithm is proposed the episode-based version of REPS is presented in Kupcsik et al. (2013) which is equivalent to stochastic search. Using Taylor approximations of the KL-divergence leads to the natural evolutionary strategies (NES) presented in Wierstra et al. (2014). NES uses the concept of the natural gradient, where the update is performed according to the standard gradient while simultaneously satisfying the KL-bound between subsequent search distributions.

The Covariance Matrix Adaptation-Evolutionary Strategy (CMA-ES) (Hansen, 2016) is a widely used stochastic search algorithm. It is based on well-defined heuristics to update the search distribution and can be considered state of the art in terms of sample efficiency.

A contextual version of MORE has been explored in Tangkaratt et al. (2017), where variables that do not change for a given task but vary from one task to another give the notion of context. Enabling learning from high-dimensional context variables like camera images.

Model-based methods for approximating the objective function with a local surrogate have been used in derivative free optimization Nocedal and Wright (2006). Also, local surrogate models have been used in trust region methods like Powell (2009). These methods maintain a

point estimate and a trust region around this point instead of a search distribution. The point estimate is updated by optimizing the surrogate and staying in the trust region.

Recursive estimation with methods like the Kalman Filter is extensively used in robotics (Chen, 2011), mainly to estimate the state and environment of the robot from noisy sensor measurements.

Chapter 4.

Recursive Surrogate-Modeling for MORE

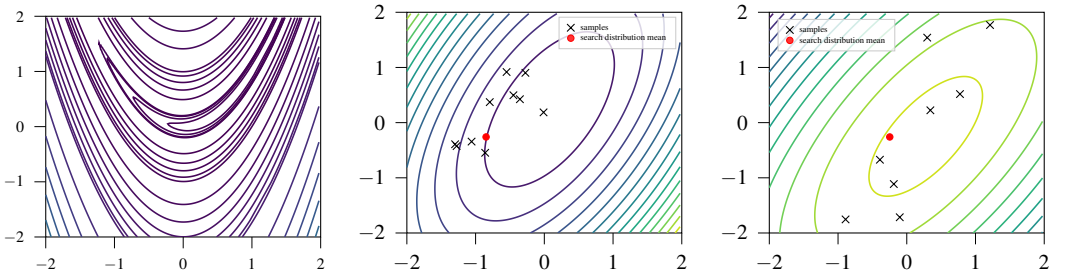
In this chapter, we will first discuss the quadratic surrogate model and motivate the idea of recursively estimating it. Then we formulate the surrogate model estimation task as a regression problem. Next we introduce a batch solution that is based on ridge regression and the Recursive Least Squares with Drift Model algorithm as our recursive approach for solving the problem.

4.1. Quadratic Surrogate Model

To compute the policy update the MORE algorithm uses a quadratic surrogate model of the objective function. The surrogate model has the following form

$$f(\mathbf{x}) \approx \hat{f}(\mathbf{x}) = \mathbf{x}^T \mathbf{R} \mathbf{x} + \mathbf{x}^T \mathbf{r} + r$$

where $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is the original objective function. The surrogate model $\hat{f}(\mathbf{x})$ has a quadratic term $\mathbf{R} \in \mathbb{R}^{n \times n}$, a linear term $\mathbf{r} \in \mathbb{R}^n$ and a scalar r. Since we assume that the search distribution is Gaussian, we need to use a quadratic model in order to derive the closed form solution for the update of the search distribution. Using a Gaussian search distribution combined with a quadratic model is sufficient as the exponential of the Gaussian is also quadratic in the parameters and a more complex model could not be exploited by a Gaussian distribution (Abdolmaleki et al., 2015). The original approach for estimating the surrogate model employs a form of Bayesian dimensionality reduction combined with linear regression. This is done from scratch in each iteration using samples and corresponding values of the objective function. This approach turns out to be very data-intensive and computationally



(a) Objective Function (b) Surrogate Model (iteration i) (c) Surrogate Model (iteration $i+1$)

Figure 4.1.: In (a) the original objective function, a 2 dimensional Rosenbrock function (Section 5.2.1) is shown. In (b) and (c) we see the search distribution mean, the surrogate model and the samples used to estimate the model from two subsequent MORE iterations.

expensive. By employing the data preprocessing techniques introduced in Section 2.6 we can instead solve the problem in the original dimension. Additionally, subsequent models are correlated due to the locality of the data, see Figure 4.1 for an example of two subsequent surrogate models. This arises from the fact that the KL-divergence is used to bound the distance between subsequent search distributions. Therefore recursive estimation techniques may be able to utilize the information contained in the previous surrogate model and thus reduce the total amount of samples needed. This also may reduce the runtime of the algorithm. Nevertheless estimating the surrogate model is a challenging task as we have to estimate $\mathcal{O}(n^2)$ many parameters while using a minimal amount of samples in each MORE Iteration.

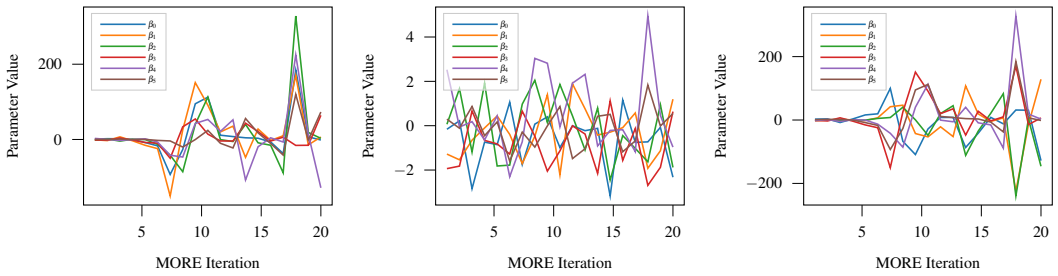
4.2. Ridge Regression

Let us now introduce a batch solution for learning the model, which we will use as a comparison to the recursive approach in our experiments. Therefore we formulate the task of estimating the surrogate model as a regression problem (4.1). For this we use a feature function $\phi(\mathbf{x})$ which returns a bias term, all linear and all quadratic terms. The dimensionality of $\phi(\mathbf{x})$ is $D = 1 + d + d(d+1)/2$, where d is the dimensionality of the parameter space.

$$y = \phi(\mathbf{x})\beta + \varepsilon \tag{4.1}$$

Our data consists of the samples and corresponding objective values $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$. To solve the regression problem we set up the design matrix $\mathbf{X}$ as depicted in equation (4.2). A row of $\mathbf{X}$ contains a 1 as the first entry for the bias term. Then the next d entries are made up of the corresponding sample $\mathbf{x}_k$ for the linear term. The final $d(d+1)/2$ entries are the lower triangular matrix of the product $\mathbf{x}_k\mathbf{x}_k^T$, which we denote by $\text{tril}(\mathbf{x}_k\mathbf{x}_k^T)$.

For the batch solution with $n \geq D$ samples we get the following system of linear equations



(a) Without Whitening (b) In Whitened Space (c) Unwhitened Parameters

Figure 4.2.: Example of using MORE with a least squares approach for surrogate-modeling with and without whitening on the 2-dimensional Rosenbrock function (Section 5.2.1). The plots show the predicted surrogate model parameters β. In (a) the parameters are estimated without whitening the samples. In (b) a whitening transformation is applied to the samples before parameter estimation and (c) shows the parameters from (b) after reversing the whitening transformation. We can see that the parameters in whitened space stay in a smaller range making the estimation task easier.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & \mathbf{x}_1 & \text{tril}(\mathbf{x}_1\mathbf{x}_1^T) \\ 1 & \mathbf{x}_2 & \text{tril}(\mathbf{x}_2\mathbf{x}_2^T) \\ \vdots & \vdots & \vdots \\ 1 & \mathbf{x}_n & \text{tril}(\mathbf{x}_n\mathbf{x}_n^T) \end{bmatrix}}_{\mathbf{X}} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_D \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}. \tag{4.2}$$

We assume that the measurement noise ε_k is zero mean Gaussian $\varepsilon_k \sim N(0, \sigma^2)$ distributed. To solve the system of linear equations in 4.2 we use a form of ridge regression (Hoerl et al., 1975).

$$\beta = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

where λ is the ridge parameter chosen as 1×10^{-12}. The solution vector β contains the surrogate model parameters $(r, \mathbf{r}, \mathbf{R})$. As the regression is done in the original problem space, which tends to be high dimensional especially for robotic tasks, we now explore how data preprocessing techniques can make the estimation task easier.

For the samples we use *Cholesky whitening* which is based on Cholesky factorization of the precision matrix $\mathbf{L}\mathbf{L}^T = \Sigma^{-1}$, which leads to the whitening matrix $\mathbf{W}^{\text{Chol}} = \mathbf{L}^T$. If the Cholesky whitening is unsuccessful due to numerical problems, we instead standardize the random vector meaning $\text{var}(\mathbf{z}) = 1$ but the correlations are not removed. In Figure 4.2 we provide an example which illustrates the effect of whitening for our parameter estimation task.

To reduce the number of samples needed per iteration we use a sample pool, which contains samples from previous iterations. To deal with data from unsmooth reward functions caused by penalties (e.g. self-collision of the robot) we normalize the reward. For this the mean and

standard deviation of the sample pool is used to compute the z-score for each objective value y as follows

$$z = \frac{y - \mu_{\text{pool}}}{\sigma_{\text{pool}}}.$$

In our robotic tasks the reward signal can have sharp jumps due to penalties (e.g. self-collision of the robot). To avoid getting stuck after receiving a penalty we additionally experimented with clipping the rewards.

4.3. Recursive Least Squares with Drift Model

Let us now explain how we use the recursive least squares (RLS) with drift model algorithm to compute the solution vector β to the regression problem. Using a recursive estimation approach we process each pair of samples and rewards $(\mathbf{x}_k, y_k)$ one at a time

$$y_k = \begin{pmatrix} 1 & \mathbf{x}_k & \text{tril}(\mathbf{x}_k \mathbf{x}_k^T) \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_D \end{pmatrix} + \varepsilon_k.$$

As the model parameters do not stay constant between MORE iterations we assume they perform a Gaussian random walk between measurements and for an arbitrary step k we get

$$\begin{aligned} p(y_k|\,\beta_k) &= \mathcal{N}(y_k|\,\mathbf{H}_k\beta_k, \sigma^2) \\ p(\beta_k|\,\beta_{k-1}) &= \mathcal{N}(\beta_k|\,\beta_{k-1}, \mathbf{Q}) \\ p(\beta_0) &= \mathcal{N}(\beta_0|\,\mathbf{m}_0, \mathbf{P}_0) \end{aligned}$$

where $\mathbf{Q}$ is the covariance of the random walk. The parameter noise is added to the covariance matrix of the parameters before each update step. Our prior $\mathbf{m}_0$ is chosen such that for the first surrogate model parameters we set the quadratic term to an identity matrix $\mathbf{R} = \mathbf{I}$ and the other terms to zero $\mathbf{r} = \mathbf{0}$ and $r = 0$. The covariance matrix is initialized as $\mathbf{P}_0 = \delta\mathbf{I}$ with the parameter δ controlling how confident we are in our prior. We want to compute the filtering distribution

$$\begin{aligned} p(\beta_k|\,y_{1:k}, \mathbf{x}_{1:k}) &\propto p(y_k, \mathbf{x}_k|\,\beta_k)\, p(\beta_k|\,y_{1:k-1}, \mathbf{x}_{1:k-1}) \\ &\propto \mathcal{N}(\beta_k|\,\mathbf{m}_k, \mathbf{P}_k), \end{aligned}$$

for this we can use the update equation of the Kalman filter (2.8). Using the model noise to increase the covariance matrix of the parameters combined with the Kalman filter update step forms the basis of Algorithm 1.

We further explored adding various data preprocessing techniques to the algorithm. First of all, like the ridge regression we also use whitening on the samples. To cope with sharp jumps in the objective function due to penalties we normalize the reward y_k. For computing the mean

and standard deviation we use the exponential moving average (EMA) (Equation (2.10)), thus for the standard score of the reward we get

$$z = \frac{y - \mu_{\text{EMA}}}{\sigma_{\text{EMA}}}.$$

By using the EMA the normalization can be computed iteratively in an on-line fashion. In addition we can use the weighting factor α of the EMA to control how fast older rewards should be discounted.

Opposed to the ridge regression we do not want to use a sample pool for the RLS and instead rely on the information contained in the previous surrogate model. Nonetheless in our experiments we did not receive good results when using only new samples without a sample pool. The predictions for the surrogate models were to inaccurate. To overcome this we tried using a warm start, meaning we process a large batch of samples in the first iteration to get an accurate prediction. The prediction can then be subsequently updated with new samples. This improved the results but it still performed worse than the batch solution. Thus we also tried using a sample pool for our recursive estimation approach. When using a sample pool we generally process the newest samples last. Additionally, we explored giving older samples a greater model noise, encoding our increased uncertainty about them. For this we simply introduced a counter for each sample, indicating how many times the sample has been used. For older samples (higher counter) we increase the model noise controlled by a weight factor γ. When using this sample weighting we increase the model noise by computing

$$\mathbf{Q} + \gamma \cdot \text{counter} \cdot \mathbf{I}$$

before adding the model noise to the covariance matrix of the parameters. Still using a sample pool is theoretically imprecise and part of future work is to explore ways to avoid it. One way to achieve this might be learning the model noise or dynamically adjusting it to the changes in the objective values.

During our research we did not find a good state transition model, therefore we only use the update step of the Kalman filter equation (2.8). We tried some momentum based approaches but did not receive any promising results. Therefore our approach is a special case of the Kalman filter equations with an Identity matrix as the state transition matrix $\mathbf{A}$. Exploring different state transition models and thereby incorporating the prediction step of the Kalman filter is part of future work.

Our final approach for recursive surrogate-modeling is summarized in Algorithm 1. The RLS with drift model algorithm and the ridge regression from section 4.2 can easily be combined with the MORE algorithm, for this only the part for learning the quadratic surrogate model $\hat{f}$ has to be changed in the original MORE (Algorithm 2).

Algorithm 1 : Recursive Least Squares with Drift Model

Input : stream of samples as rows of design matrix $\mathbf{X}_n$ and rewards y_n with $n = 1, \ldots, N$, $\mathbf{Q}$ model noise, σ^2 measurement noise

Initialization: $\mathbf{m}_0 = (0, \mathbf{0}, \mathbf{I})$, $\mathbf{P}_0 = \delta\mathbf{I}$

for $n = 1, ..., N$ **do**

  begin Data Preprocessing

    Use whitening on $\mathbf{X}_n$

    Compute exponential moving average and normalize y_n

  end

  $\mathbf{P}_n^- = \mathbf{P}_{n-1} + \mathbf{Q}$ *// Add the model noise*

  begin Update step

    $S_n = \mathbf{X}_n \, \mathbf{P}_n^- \, \mathbf{X}_n^T + \sigma^2$

    $\mathbf{K}_n = \mathbf{P}_n^- \, \mathbf{X}_n^T \, S_n^{-1}$

    $\mathbf{m}_n = \mathbf{m}_{n-1} + \mathbf{K}_n [y_n - \mathbf{X}_n \mathbf{m}_{n-1}]$

    $\mathbf{P}_n = \mathbf{P}_n^- - \mathbf{K}_n \, S_n \, \mathbf{K}_n^T$

  end

end

Reverse whitening transformation for $\mathbf{m}_N$

return $\mathbf{m}_N$

Chapter 5.

Evaluation

In this chapter we first describe the setup for our experiments. Then we introduce the Rosenbrock function, the planar reaching task and the ball-in-a-cup task. For each problem we present the best results we could find with our algorithm and compare it to the batch solution and the original MORE approach.

5.1. Setup

All algorithms were implemented in python using NumPy[1]. The optimization algorithm for MORE is Low-storage BFGS from NLopt[2]. The clusterwork2 (cw2) framework, developed at the Autonomous Learning Robots (ALR) Lab, was used for running experiments in a convenient way. The experiments were conducted on a machine with two 8-core AMD Ryzen 2700X processors clocked at 2.6 GHz and 31GB of RAM.

5.2. Experiments

The focus in our experiments is on the sample efficiency of the MORE algorithm. While the original MORE algorithm achieves state-of-the-art results in terms of optimization accuracy the surrogate model estimation is data-intensive. Therefore the metric we focus on is sample

[1] https://numpy.org/
[2] https://nlopt.readthedocs.io/en/latest/

efficiency for which we use the heuristic $s = 4 + \lfloor 3\log(n) \rfloor$ of the CMA-ES algorithm (Hansen, 2016) as a benchmark, with n being the dimension of the problem and s the number of samples drawn from the search distribution in each iteration.

We compare our recursive least squares (RLS) approach to the ridge regression (RR) and the original MORE algorithm, which uses Bayesian dimensionality reduction approach combined with linear regression (BLR). Generally, we use the CMA-ES heuristic to determine the number of samples. As we are working with a stochastic search algorithm we do 10 runs with each approach and calculate the *arithmetic* mean.

5.2.1. Test Functions for Optimization

Test functions provide artificial landscapes to evaluate the performance of optimization algorithms. They can be used to assess robustness, convergence speed and general performance of the algorithms (Molga and Smutnicki, 2005).

The Rosenbrock function (Figure 5.1) is uni-modal and convex. It has one global optimum which is inside a parabolic shaped flat valley. It is usually easy to find the valley, however finding the global optimum is particularly difficult, which makes this function popular for testing the performance of algorithms. It is defined as

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2].$$

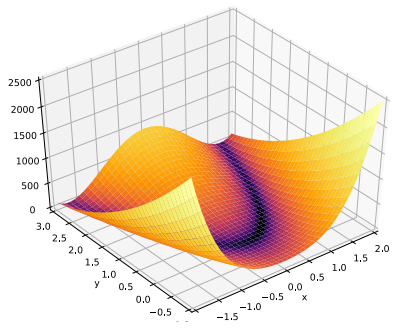


Figure 5.1.: 2D Rosenbrock function

The function has a global minimum at zero ($f(\mathbf{x}) = 0$). In our experiments we initialize the mean of the search distributions randomly. We test our algorithm on the 15 dimensional Rosenbrock function, the results are shown in Figure 5.2. For these results RLS uses 7 samples per iteration and a pool of size 100, whereas the ridge regression uses 12 samples per iteration and a pool of size 204. The BLR approach only worked with using 30 samples per iteration and a pool of size 150. The RLS approach outperforms the other methods in terms of total samples needed for convergence.

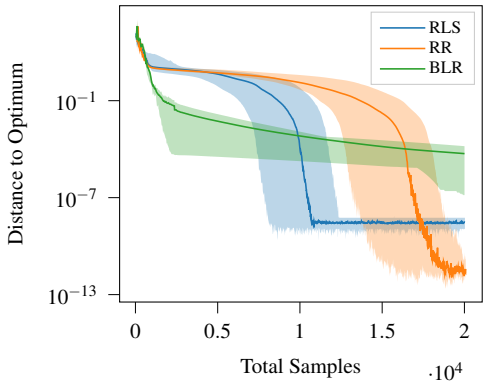


Figure 5.2.: This figure shows the mean of 10 runs for RLS, ridge regression (RR) and the original MORE algorithm, which uses Bayesian Linear Regression (BLR), on 15 dimensional Rosenbrock.

5.2.2. Planar Reaching Task

Policy search methods enable us to work with parameterized policies, therefore we can use a policy representation that is relevant to a given task. In robotics Dynamic Movement Primitives (DMPs) first introduced in Ijspeert et al. (2002) are commonly used for movement tasks. Formalized as second-order dynamic systems, DMPs offer a compact representation for movements, allowing to choose between a rhythmic and a discrete movement.

For our task we use a 5 link robot with DMPs as the underlying control policy. The end-effector of the robot has to reach a via-point $v_{100} = [1,1]$ at time step 100 and at the final time step $T = 200$ the via-point $v_{200} = [5,0]$. The reward is given by a quadratic cost term for the two via-points, for self-collision a penalty of 100 is given. As the the via-points are defined in end effector space the objective function is highly non-quadratic in the parameters. Using 5 basis functions per degree of freedom for the DMPs results in a 25 dimensional parameter vector. In Figure 5.3 we see the average reward and the resulting movement from one RLS run. The ridge regression had more problems with avoiding a self-collision than the other two approaches. The RLS algorithm performs slightly better than BLR and RR, but overall these the improvement seems not very significant as we have to take the underlying stochastic nature into account. But nonetheless we could show that MORE with recursive surrogate-modeling is able to learn the reaching task.

5.2.3. Ball in a Cup

The motor skill game Ball-in-a-Cup consists of small cup that has a string with a ball attached to it. Initially the ball hangs down vertically in a resting position. See Figure 5.4 for a picture of the setup in the simulation framework. The goal is to toss the ball up and catch it inside the cup. The reward signal is based on calculating the distance d from the center of the

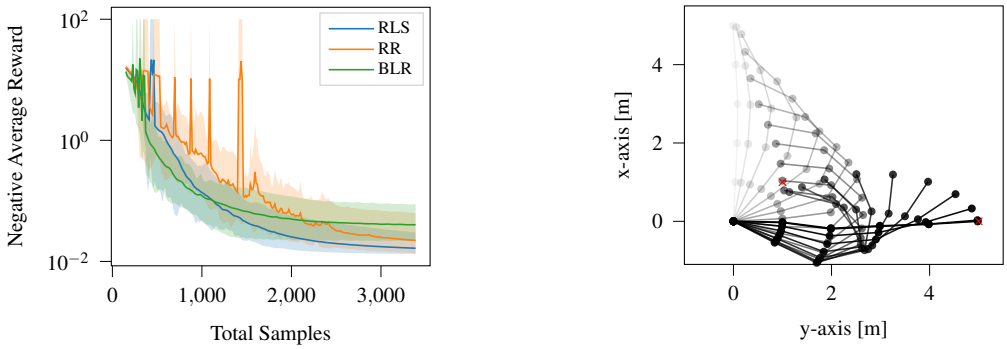


Figure 5.3.: Left: Results on via-point reaching task using the recursive least squares (RLS), the ridge regression (RR) and the original approach (BLR). Right: Resulting movement for via-point reaching task of the RLS algorithm. The via-points are indicated by the red crosses. The postures of the resulting motion are shown as overlay. Darker postures are later in time.

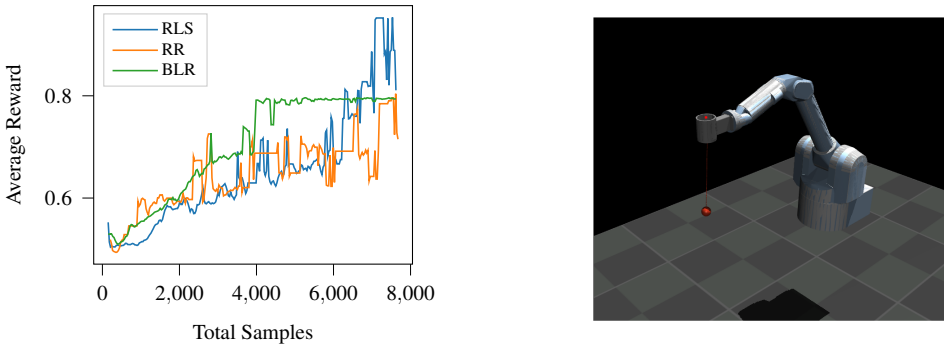


Figure 5.4.: Left: Results on ball-in-a-cup task using the recursive least squares (RLS), the ridge regression (RR) and the original approach (BLR), using the mean of 5 runs. Right: Picture of the ball-in-a-cup task in the simulation framework.

cup to the ball. When the ball lands directly in the center of the cup, the distance is 0 and through the transformation $\exp(-d^2)$ it yields the highest possible reward of 1. We ran a simulation with the Barret WAM robot arm with three degree of freedom (DoF) using the MuJoCo[3] physics engine. Our results are shown in Figure 5.4. In our remaining time for the thesis we could only complete 5 runs for each approach as the samples have to be evaluated using the mujoco framework which produces long runtimes. Therefore the results should be considered preliminary. The BLR seems to converge prematurely, though this may be due to a suboptimal entropy constraint parameter. Still, we can see that RLS is succesfull in learning the ball-in-a-cup task while the ridge regression struggles with finding a good solution.

[3] http://www.mujoco.org/

Chapter 6.

Conclusion and Future Work

6.1. Conclusion

In this work, we have presented an approach for combining the MORE algorithm (Abdolmaleki et al., 2015) with recursive surrogate-modeling using the RLS algorithm with drift model (Algorithm 1). The main goal of this approach is to improve the sample efficiency of MORE, as real world samples are expensive and previously learning the surrogate model has been data-intensive. We tested our algorithm on the Rosenbrock function and a two simulated robotic task. While we were able to clearly improve sample efficiency on the Rosenbrock function, we did not manage to achieve similar improvements on the robotic tasks. Nevertheless we could successfully learn the planar reaching task and the ball-in-a-cup task, which shows that the recursive surrogate-modeling approach has potential to reach and improve the current state-of-the art in terms of sample efficiency. Our approach leaves much room for improvement in terms of finding the optimal combination of hyperparameters and different data preprocessing techniques. With more research the improved sample efficiency observed on the optimization test function may be transferred to complex robotic tasks.

6.2. Future Work

This section lists possibilities for further improvements of using recursive surrogate modeling for the MORE algorithm.

Using a State Transition Model. As a next step finding a fitting state transition matrix **A** and using the prediction step of the Kalman filter may improve results. The author had little experience with Kalman filters prior to this project, hence work on optimizing the algorithm for recursive estimation may lead to significant improvements.

Re-evaluating, optimizing hyperparameters. One disadvantage of MORE is the number of parameters, our recursive surrogate-modeling algorithm introduced additional parameters. As we relied mainly on manual and grid search for hyperparameters a more sophisticated approach could yield considerable improvements. Though the stochastic nature of the algorithm makes comparing different configurations difficult and a more thorough evaluation could reveal new insights.

Learning the Model Noise. When using no pool the model noise parameter stays constant for all MORE iterations. Although we optimized this constant value it makes more sense to dynamically adjust the model noise in some way related to the changes values of the objective function. In the future exploring different techniques for learning the model drift may proof successful in increasing the performance.

Testing on Complex Robot Learning Tasks We only conducted experiments on a simple simulated robot task. The obvious next step is to benchmark the algorithm on more complicated dimensional robotic tasks. This might reveal shortcomings of the approach for dealing with unsmooth objective functions and fulfilling the goal of safe exploration.

Bibliography

A. Abdolmaleki, R. Lioutikov, J. R. Peters, N. Lau, L. Pualo Reis, and G. Neumann. Model-based relative entropy stochastic search. *Advances in Neural Information Processing Systems*, 28:3537–3545, 2015.

R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

S. Chen. Kalman filter for robot vision: a survey. *IEEE Transactions on Industrial Electronics*, 59(11):4409–4420, 2011.

M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

M. P. Deisenroth, G. Neumann, and J. Peters. *A survey on policy search for robotics*. now publishers, 2013.

T. Finch. Incremental calculation of weighted mean and variance. *University of Cambridge*, 4 (11-5):41–42, 2009.

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*. CRC press, 2013.

N. Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

A. E. Hoerl, R. W. Kannard, and K. F. Baldwin. Ridge regression: some simulations. *Communications in Statistics-Theory and Methods*, 4(2):105–123, 1975.

A. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. *Advances in neural information processing systems*, 15:1547–1554, 2002.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, Mar. 1960.

B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on automation science and engineering*, 12(2):398–409, 2015.

A. Kessy, A. Lewin, and K. Strimmer. Optimal whitening and decorrelation. *The American Statistician*, 72(4):309–314, 2018.

J. Kober and J. Peters. Policy search for motor primitives in robotics. *Machine learning*, 84 (1-2):171–203, 2011.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

A. G. Kupcsik, M. P. Deisenroth, J. Peters, G. Neumann, et al. Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013*, pages 1401–1407, 2013.

W. M. Mendenhall and T. L. Sincich. *Statistics for Engineering and the Sciences*. CRC Press, 2016.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

M. Molga and C. Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 101:48, 2005.

A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

T. Osa, J. Pajarinen, G. Neumann, J. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 7(1-2):1–179, Mar. 2018.

J. Peters, K. Mülling, and Y. Altun. Relative entropy policy search. In *AAAI*, volume 10, pages 1607–1612. Atlanta, 2010.

M. J. Powell. The bobyqa algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, pages 26–46, 2009.

S. Russell. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, 1998.

S. Särkkä. *Bayesian filtering and smoothing*, volume 3. Cambridge University Press, 2013.

S. Schaal. The new robotics—towards human-centered machines. *HFSP journal*, 1(2): 115–126, 2007.

S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In *Robotics research. the eleventh international symposium*, pages 561–572. Springer, 2005.

J. G. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. *Advances in neural information processing systems*, pages 1047–1053, 1997.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22, 1992.

V. Tangkaratt, H. van Hoof, S. Parisi, G. Neumann, J. Peters, and M. Sugiyama. Policy search with high-dimensional context variables. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

S. Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Appendix A.

Appendix

A.1. MORE: Equations

Forming the Lagrangian for the constraint optimization problem we get

$$\mathfrak{L}(\pi,\eta,\omega) = \int \pi(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} + \eta \left(\varepsilon - \int \pi(\mathbf{x}) \log \frac{\pi(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} \right) - \omega \left(\beta + \int \pi(\mathbf{x}) \log(\pi(\mathbf{x})) d\mathbf{x} \right)$$

Optimizing the Lagrangian by computing the derivative with respect to the mean and covariance matrix yields

$$\pi(\mathbf{x}) \propto q(\mathbf{x})^{\eta/(\eta+\omega)} \exp\left(\frac{f(\mathbf{x})}{\eta+\omega} \right)$$

the solution depends on the quadratic and linear term of the surrogate model and the Lagrangian multipliers η and ω. These in turn can be obtained by minimizing the dual function:

$$g(\eta,\omega) = \eta\varepsilon - \omega\beta + (\eta-\omega)\log\left(\int q(\mathbf{x})^{\frac{\eta}{\eta+\omega}} \exp\left(\frac{f(\mathbf{x})}{\eta+\omega} \right) d\mathbf{x} \right) \tag{A.1}$$

Assuming we are given a quadratic surrogate model

$$f(\mathbf{x}) \approx \hat{f}(\mathbf{x}) = \mathbf{x}^T \mathbf{R} \mathbf{x} + \mathbf{x}^T \mathbf{r} + r_0$$

we can solve the dual function in closed form.

$$g(\eta,\omega) = \eta\varepsilon - \beta\omega + \frac{1}{2}\left(\mathbf{f}^T\mathbf{F}\mathbf{f} - \eta\mathbf{b}^T\mathbf{Q}^{-1}\mathbf{b} - \eta\log|2\pi\mathbf{Q}|p + (\eta+\omega)\log|2\pi(\eta+\omega)\mathbf{F}|\right)$$

with $\mathbf{F} = (\eta\Sigma^{-1} - 2\mathbf{R})^{-1}$ and $\mathbf{f} = \eta\Sigma^{-1}\mu + \mathbf{r}$

A.2. Gaussian Distribution

A random variable $x \in \mathbb{R}^n$ has a Gaussian distribution with mean $m \in \mathbb{R}^n$ and covariance $P \in \mathbb{R}^{n\times n}$ if its probability density has the form

$$\mathcal{N}(x|m,P) = \frac{1}{(2\pi)^{n/2}|P|^{1/2}}\exp\left(-\frac{1}{2}(x-m)^T P^{-1}(x-m)\right)$$

where $|P|$ is the determinant of the matrix P.

A.3. MORE Algorithm

Algorithm 2 : MORE

Input : Parameters ε and β, initial search distribution π
K number of iterations, N samples per iteration
for $k = 1,...,K$ **do**
  for $n = 1,...,N$ **do**
    Draw sample $\mathbf{x}_n \sim \pi$
    Evaluate $\mathbf{x}_n$ on objective function $f(\mathbf{x}_n) = y_n$
  end
  Learn the quadratic model $\hat{f}$
  Solve $\text{argmin}_{\eta>0,\omega>0}\, g(\eta,\omega)$ using Equation (A.1)
  Update the search distribution π using Equation (2.2)
end