Learning of Non-Parametric Control Policies with High-Dimensional State Features

Herke van Hoof[‡] **Jan Peters**[‡*] **Gerhard Neumann**[‡]
[‡]TU Darmstadt, Computer Science Department. [*]MPI for Intelligent Systems.

Abstract

Learning complex control policies from high-dimensional sensory input is a challenge for reinforcement learning algorithms. Kernel methods that approximate values functions or transition models can address this problem. Yet, many current approaches rely on instable greedy maximization. In this paper, we develop a policy search algorithm that integrates robust policy updates and kernel embeddings. Our method can learn non-parametric control policies for infinite horizon continuous MDPs with high-dimensional sensory representations. We show that our method outperforms related approaches, and that our algorithm can learn an underpowered swing-up task task directly from high-dimensional image data.

1 Introduction

Learning continuous valued control policies directly from high-dimensional sensory input presents a major obstacle to applying reinforcement learning (RL) methods effectively in realistic settings. Current approaches for continuous domains typically rely on human-designed features for value function approximations or specialized parametric policies [Baird and Moore, 1999, Peters et al., 2010], which is impractical for high dimensional sensory inputs.

Furthermore, many methods require a transition model, which, in the general case, is unknown. A recent approach [Grünewälder et al., 2012b] embeds the transition distributions in a reproducing kernel Hilbert space (RKHS). Rather than estimating densities, this technique directly estimates expected values efficiently [Song et al., 2013, Grünewälder et al., 2012b]. RKHS have the additional benefit that they implicitly define a (possibly infinite) features space, reducing the effort of defining features.

Such RKHS embeddings have been successfully used in value-function methods [Grünewälder et al., 2012b, Nishiyama et al., 2012]. These methods update the policy greedily with respect to the learned value function, which can cause instabilities in the learning progress as the learned value function is only an approximation.

Such instabilities can be avoided by bounding the information loss between subsequent state-action distributions [Peters et al., 2010]. In discrete finite horizon problems, such a bound has been proven to have optimal regret in an adversarial MDP [Zimin and Neu, 2013]. Empirically, it yields good results in real-world continuous MDPs [Kupcsik et al., 2013, Lioutikov et al., 2014, Daniel et al., 2013]. However, methods based on this insight have so far required linear approximation using hand-crafted features, and are not applicable to general non-linear MDPs.

In this paper, we employ RKHS embeddings to find bounded policy updates according to the information-theoretic objective proposed in [Peters et al., 2010]. This method contributes *a more stable, effective learning progress* for non-parametric reinforcement learning. Simultaneously, it *avoids the use of hand-crafted features in non-linear continuous systems,* by extending the problem formulation to infinite feature spaces. Furthermore, our method *allows using non-parametric policies*, which is not possible with most policy gradients updates.

The resulting robust RL method performs well in continuous state-action spaces with high-dimensional sensory representations. In our experiments, we show that our method outperforms relevant baselines on a reaching task and an underpowered swing-up task. We also show it is applicable to a task with high dimensional pixel images state representation.

Appearing in Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS) 2015, San Diego, CA, USA. JMLR: W&CP volume 38. Copyright 2015 by the authors.

1.1 Notation and Background

In a Markov decision process (MDP), an agent in state $\mathbf{s}$ selects an action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$ according to a (possibly stochastic) policy π and receives a reward $\mathcal{R}_\mathbf{s}^\mathbf{a} \in \mathbb{R}$. We will assume continuous state-action spaces: $\mathbf{s} \in \mathcal{S} = \mathbb{R}^{D_\mathrm{s}}$, $\mathbf{a} \in \mathcal{A} = \mathbb{R}^{D_\mathrm{a}}$. If the Markov decision process is ergodic, for each policy π, there exists a stationary distribution $\mu_\pi(\mathbf{s})$ such that $\int_\mathcal{S}\int_\mathcal{A} \mathcal{P}_{\mathbf{ss}'}^\mathbf{a}\pi(\mathbf{a}|\mathbf{s})\mu_\pi(s)\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} = \mu_\pi(\mathbf{s}')$, where $\mathcal{P}_{\mathbf{ss}'}^\mathbf{a} = Pr(\mathbf{s}'|\mathbf{a},\mathbf{s})$. The goal of a reinforcement learning agent is to choose a policy such that the joint state-action distribution $p(\mathbf{s},\mathbf{a}) = \mu_\pi(\mathbf{s})\pi(\mathbf{a}|\mathbf{s})$ maximizes the average reward $J(\pi) = \int_\mathcal{S}\int_\mathcal{A} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{R}_\mathbf{s}^\mathbf{a}\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}$.

To avoid overly greedy optimization and instabilities in the learning process, the Kullback-Leibler (KL) divergence of a sampling distribution $q(\mathbf{s},\mathbf{a})$ from the state-action distribution $p(\mathbf{s},\mathbf{a})$ can be bounded [Peters et al., 2010], leading to the optimization problem

$$\max_{\pi,\mu_\pi} J(\pi) = \max_{\pi,\mu_\pi} \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{R}_\mathbf{s}^\mathbf{a}\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}, \quad (1)$$

$$s.t. \quad \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathrm{d}\mathbf{s}\mathrm{d}\mathbf{a} = 1, \quad (2)$$

$$\forall s' \quad \iint_{\mathcal{S}\times\mathcal{A}} \mathcal{P}_{\mathbf{ss}'}^\mathbf{a}\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} = \mu_\pi(\mathbf{s}'), \quad (3)$$

$$\mathrm{KL}(\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})||q(\mathbf{s},\mathbf{a})) \leq \epsilon, \quad (4)$$

where Eqs. (1-3) specify the general reinforcement learning objective (1) with the constraints that $\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})$ is a distribution (Eq. 2) and μ_π is the stationary distribution under $\pi(\mathbf{a}|\mathbf{s})$ (Eq. 3). Equation (4) specifies the additional bound on the KL divergence, where

$$\mathrm{KL}(p(x)||q(x)) = \int p(x)\log(p(x)/q(x))\mathrm{d}x.$$

Reference distribution q is usually set to the state-action distribution induced by previous policies, with the initial explorative policy a wide, uninformed distribution. As learning progresses, the policy typically slowly converges towards a deterministic policy.

As we typically know $\mathcal{R}_\mathbf{s}^\mathbf{a}$ *only* for a sequence of samples $\{(\mathbf{s}_1,\mathbf{a}_1),\ldots(\mathbf{s}_n,\mathbf{a}_n)\}$, the expectations are approximated by the sample mean of their argument. The solution to the optimization problem obtained through Lagrangian optimization[1] is given by

$$\pi(\mathbf{a}_i|\mathbf{s}_i)\mu_\pi(\mathbf{s}_i) \propto q(\mathbf{s}_i,\mathbf{a}_i)\exp\left(\frac{\delta(\mathbf{s}_i,\mathbf{a}_i,V)}{\eta}\right), \text{with}$$
$$\delta(\mathbf{s},\mathbf{a},V) = \mathcal{R}_\mathbf{s}^\mathbf{a} + \mathbb{E}_{\mathbf{s}'}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}] - V(\mathbf{s}) \quad (5)$$

[1]The corresponding derivation is given in Appendix 1.

where $V(\mathbf{s})$ and η denote Lagrangian multipliers, and δ denotes the Bellman error [Peters et al., 2010]. These multipliers are obtained through minimization of the dual function

$$g(\eta,V) = \eta\epsilon + \eta\log\left(\sum_{i=1}^n \frac{1}{n}\exp\left(\frac{\delta(\mathbf{s}_i,\mathbf{a}_i,V)}{\eta}\right)\right),$$
$$(\mathbf{s}_i,\mathbf{a}_i) \sim q(\mathbf{s},\mathbf{a}) \quad (6)$$

The Lagrangian multiplier $V(\mathbf{s})$ is a function of $\mathbf{s}$ and resembles a value function. To calculate the Bellman error δ, the transition distribution is required. As this distribution is generally not known, δ needs to be approximated. The dual function (6) depends implicitly on reference distribution q through the samples.

1.2 Problem Statement

In this paper, we aim at developing a method applicable in continuous state-action MDPs with high-dimensional state representations. We assume hand-coded feature functions and parametric policies are not available and the transition and reward models of the MDP are unknown. Furthermore, we will concentrate on infinite-horizon problems.

2 Using Conditional Embeddings to solve Continuous MDPs

In this section, we explain the methods needed to stably learn non-parametric controllers. First, we show how to use conditional embeddings to approximate the expectated values in Eqs. (1)-(4) and we show how to solve this optimization problem with respect to a non-linear function V. After that, we discuss how to relax the assumption of ergodicity of the MDP by transforming the average reward MDP in a discounted reward MDP. Solving the optimization problem results in a new optimal policy that is, however, only defined on the current set of samples. Therefore, we discuss how the sample-based optimal policy can be generalized to the entire state space. Finally, we will discuss how to set the hyper-parameters of the different steps of our method without manual tuning.

2.1 Solving the Dual Problem

To solve Eqs. (1)-(4), we need to minimize the dual problem in (6): $(\mu^*,V^*) = \arg\max g(\eta,V)$. We will assume that $V^* \in \mathcal{F}$ for some reproducing kernel Hilbert space (RKHS) $\mathcal{F}$ with kernel k_s, i.e., we assume V^* is of the form

$$V^* = \sum_{\tilde{\mathbf{s}}\in\tilde{\mathcal{S}}} \alpha_{\tilde{\mathbf{s}}} k_\mathrm{s}(\tilde{\mathbf{s}},\cdot), \quad (7)$$

for some set $\tilde{\mathcal{S}}$ and scalars α. The kernel k_s implicitly defines a (possibly infinite dimensional) feature map $\boldsymbol{\phi}(\mathbf{s}) = k_\mathrm{s}(\mathbf{s}, \cdot)$. The implicit definition has the advantage that we do not need to explicitly specify a feature vector for V^*. Kernels are in general easier to define than feature vectors as the complexity of V^* can grow with the amount of training data.

Embedding the transition model. Since the transition model $\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}$ is unknown, we need to approximate $\mathbb{E}_{\mathbf{s}'}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}]$. To do so, we embed the conditional $\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}$ in the RKHS $\mathcal{F}$, i.e., we represent $\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}$ by the expected implicit features $\boldsymbol{\mu}_{\mathbf{s}'|\mathbf{s},\mathbf{a}} = \mathbb{E}_{\mathbf{s}'}[\boldsymbol{\phi}(\mathbf{s}')|\mathbf{s},\mathbf{a}]$. Using such embeddings avoids estimating the joint density and leads to good results even for high-dimensional data [Song et al., 2013]. They also render calculations of expected values over a function in $\mathcal{F}$ straightfoward without numerical integration [Song et al., 2013]. In order to learn the conditional operator, we will use a kernel over the state-action space of the form $\boldsymbol{\psi}(\mathbf{s},\mathbf{a}) = k_\mathrm{s}(\mathbf{s},\cdot)k_\mathrm{a}(\mathbf{a},\cdot)$. Given a sample $\{(\mathbf{s}_1,\mathbf{a}_1,\mathbf{s}_1'),\ldots,(\mathbf{s}_n,\mathbf{a}_n,\mathbf{s}_n')\}$, the empirical conditional embedding is defined as

$$\hat{\boldsymbol{\mu}}_{S'|s,a} = \hat{C}_{S'|S,A}\boldsymbol{\psi}(\mathbf{s},\mathbf{a}) = \sum_{i=1}^{n} \beta_i(\mathbf{s},\mathbf{a})\boldsymbol{\phi}(\mathbf{s}_i'), \qquad (8)$$

$$\hat{C}_{S'|S,A} = \boldsymbol{\Phi}(\mathbf{K}_\mathrm{sa} + \lambda\mathbf{I})^{-1}\boldsymbol{\Psi}^T, \qquad (9)$$

where $\hat{C}_{S'|S,A}$ is a learned conditional operator that allows the computation of embedding strengths $\boldsymbol{\beta}(\mathbf{s}_i,\mathbf{a}_i) = (\mathbf{K}_\mathrm{sa} + \lambda I)^{-1}\mathbf{k}_\mathrm{sa}(\mathbf{s}_i,\mathbf{a}_i)$ [Song et al., 2013, Grünewälder et al., 2012a,b]. In this equation, $\boldsymbol{\Psi} = [\boldsymbol{\psi}(\mathbf{s}_1,\mathbf{a}_1),\ldots,\boldsymbol{\psi}(\mathbf{s}_n,\mathbf{a}_n)]$, $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\mathbf{s}_1'),\ldots,\boldsymbol{\phi}(\mathbf{s}_n')]$, $\mathbf{K}_\mathrm{sa} = \boldsymbol{\Psi}^T\boldsymbol{\Psi}$ and $\mathbf{k}_\mathrm{sa}(\mathbf{s},\mathbf{a}) = \boldsymbol{\Psi}^T\boldsymbol{\psi}(\mathbf{s},\mathbf{a})$.[2]

Functional form of V. Since k_s is a reproducing kernel and $V \in \mathcal{F}$, the expected value of V can be approximated using the embedded distribution [Song et al., 2013, Grünewälder et al., 2012a,b], i.e.,

$$\mathbb{E}_{\mathbf{s}'}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}] = \left\langle V, \hat{\boldsymbol{\mu}}_{S'|s,a} \right\rangle_{\mathcal{F}} = \sum_{i=1}^{n} \beta_i(\mathbf{s},\mathbf{a})V(\mathbf{s}_i').$$

In the dual g (Eq. 6), V is now only evaluated at sampled s_i and s_i'. Since we assumed $V \in \mathcal{F}$, the generalized representer theorem [Schölkopf et al., 2001] tells us that there is at least one optimum of the form (7) with $\tilde{\mathcal{S}}$ the set of sampled states[3]. Consequently, $\mathbb{E}_{\mathbf{s}'|\mathbf{s},\mathbf{a}}[V(\mathbf{s}')] - V(\mathbf{s}) = \boldsymbol{\alpha}^T\tilde{\mathbf{K}}_\mathrm{s}\boldsymbol{\beta}(\mathbf{s},\mathbf{a}) - \boldsymbol{\alpha}^T\mathbf{k}_{\mathrm{s}(\mathbf{s})}$, where $\tilde{\mathbf{K}}_\mathrm{s}$ is the Gram matrix with entries $[\tilde{\mathbf{K}}_\mathrm{s}]_{ji} = k_\mathrm{s}(\tilde{\mathbf{s}}_j,\mathbf{s}_i')$, and $[\mathbf{k}_s(\mathbf{s})]_j = k_\mathrm{s}(\tilde{\mathbf{s}}_j,\mathbf{s})$.

[2]This means $[\mathbf{K}_\mathrm{sa}]_{ij} = k_\mathrm{s}(\mathbf{s}_i,\mathbf{s}_j)k_\mathrm{a}(\mathbf{a}_i,\mathbf{a}_j)$, $[\mathbf{k}_\mathrm{sa}(\mathbf{s},\mathbf{a})]_i = k_\mathrm{s}(\mathbf{s}_i,\mathbf{s})k_\mathrm{a}(\mathbf{s}_i,\mathbf{a})$.

[3]A sketch of the proof following [Schölkopf et al., 2001] is given in Appendix 4.

Finding a numerical solution. The dual problem can now be stated in terms of η and $\boldsymbol{\alpha}$, with

$$g(\eta,\boldsymbol{\alpha}) = \eta\epsilon + \eta\log\left(\sum_{i=1}^{n}\frac{1}{n}\exp\left(\frac{\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})}{\eta}\right)\right), \quad (10)$$

$$\delta(\mathbf{s},\mathbf{a},\boldsymbol{\alpha}) = \mathcal{R}_\mathbf{s}^\mathbf{a} + \boldsymbol{\alpha}^T\left(\tilde{\mathbf{K}}_\mathrm{s}\boldsymbol{\beta}(\mathbf{s},\mathbf{a}) - \mathbf{k}_\mathrm{s}(\mathbf{s})\right), \qquad (11)$$

This objective is convex in $\boldsymbol{\alpha}$. Since the analytic gradient and Hessian for this objective are known[4], we employ second order optimization methods to find the optimal η and $\boldsymbol{\alpha}$. We employ an iterative optimization scheme similar to the one described by Lioutikov et al. [2014], that is, we sequentially optimize for either η or $\boldsymbol{\alpha}$ until the constraints are fulfilled within an acceptable tolerance.

If we choose kernels $\mathbf{k}_\mathrm{s}(\mathbf{s}_i,\mathbf{s}_j) = \tilde{\phi}(\mathbf{s}_i)^T\tilde{\phi}(\mathbf{s}_j)$ and $\mathbf{k}_\mathrm{sa}((\mathbf{s}_i,\mathbf{a}_i),(\mathbf{s}_j,\mathbf{a}_j)) = \mathbb{I}((\mathbf{s}_i,\mathbf{a}_i) = (\mathbf{s}_j,\mathbf{a}_j))$, we obtain the original REPS formulation by Peters et al. [2010] as a special case. In these equations, $\mathbb{I}$ is the indicator function and $\tilde{\phi}$ is a set of hand-crafted features.

2.2 Ensuring a Stationary Distribution

The REPS formulation [Peters et al., 2010] assumes the existence of a stationary distribution. However, not all MDPs have a stationary distribution for every policy π (e.g., random walks). Steady-state behavior might not be realizable for real systems that need to be started and stopped; and transient behavior, such as the swing-up of a pendulum, might be of greater interest than steady-state behavior.

We can ensure the system has a stationary distribution that includes such transients by resetting the system with a probability $1-\gamma$ at each time step. The system is then set to a state from the initial state distribution $p_1(\mathbf{s})$. In this case, the expected value of V at the next time step is given by

$$\begin{aligned}\mathbb{E}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}] &= \int_{\mathcal{S}} \gamma\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}V(\mathbf{s}') + (1-\gamma)p_1(\mathbf{s}')V(\mathbf{s}')\mathrm{d}\mathbf{s}' \\ &= \gamma\boldsymbol{\alpha}^T\tilde{\mathbf{K}}_\mathrm{s}\boldsymbol{\beta} + (1-\gamma)\hat{\boldsymbol{\mu}}_{S1}, \qquad (12)\end{aligned}$$

where $\hat{\boldsymbol{\mu}}_{S1}$ is the empirical (observed) embedding of the initial state distribution and $\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}$ are the transition probabilities of the original MDP.

This reset procedure enables learning by removing the impracticable requirement of infinite rollout length. In this way, we obtain a discounted setting similar to that used in RL methods that employ discount factors.

[4]The dual and its partial derivatives and Hessians are given in Appendix 2.

2.3 Generalizing the Sample-Based Policy

The parameters resulting from the optimization, η and $\boldsymbol{\alpha}$, can be inserted back in (10) to yield the desired probabilities $\{p(\mathbf{s}_1, \mathbf{a}_1), \dots, p(\mathbf{s}_n, \mathbf{a}_n)\}$ at the sampled $(\mathbf{s}, \mathbf{a})$ pairs. Conditioning on the current state yields the policy to be followed in the next iteration. However, since states and actions are continuous, we need to generalize from these weighted samples to nearby data points. To this end, we fit the parameters Ω of a generalizing stochastic policy $\tilde{\pi}(\mathbf{a}|\mathbf{s}; \Omega)$ by minimizing the expected Kullback-Leibler divergence $\mathbb{E}_{\mathbf{s}}\left[\mathrm{KL}(\pi(\mathbf{a}|\mathbf{s})||\tilde{\pi}(\mathbf{a}|\mathbf{s}))\right]$ of the generalizing policy from the sample-based policy $\pi(\mathbf{a}|\mathbf{s})$[5]. This minimization results in a weighted maximum likelihood estimate of Ω, with weights $w_i = p(\mathbf{s}_i, \mathbf{a}_i)/q(\mathbf{s}_i, \mathbf{a}_i)$.

The advantage of this reduction to a weighted maximum likelihood estimate is that many types of policies, including non-parametric policies, can be fitted. Due to its flexibility and its good performance on many practical problems, we use Gaussian process (GP) policies $\tilde{\pi}$ in our experiments. We use the cost-sensitive GPs introduced in the cost-regularized kernel regression (CrKR) algorithm [Kober et al., 2011]. CrKR modulates the regularization parameter with the inverse of the desirability of each data point, which we define to be the normalized weights $w_i / \max_i w_i$ obtained for each sample. The new policy

$$\tilde{\pi}(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mu(\mathbf{s}), \sigma^2(\mathbf{s})), \;\; \mu(s) = \mathbf{k}_{\mathrm{s}}(\mathbf{s})^T (\mathbf{K}_{\mathrm{s}} + \lambda \mathbf{D})^{-1} \mathbf{A},$$

$$\sigma^2(\mathbf{s}) = k + \lambda - \mathbf{k}_{\mathrm{s}}(\mathbf{s})^T (\mathbf{K}_{\mathrm{s}} + \lambda \mathbf{D})^{-1} \mathbf{k}_{\mathrm{s}}(\mathbf{s}),$$

where $k = k(\mathbf{s}, \mathbf{s})$, $\mathbf{k}_{\mathrm{s}}(\mathbf{s}) = \boldsymbol{\phi}(\mathbf{s})^T \boldsymbol{\Phi}$, $\mathbf{K}_{\mathrm{s}} = \boldsymbol{\Phi}^T \boldsymbol{\Phi}$, $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n]^T$, $\mathbf{D}$ is a diagonal matrix with $\mathbf{D}_{ii} = (w_i / \max_i w_i)^{-1}$, and λ is a regularization parameter that is set with other free kernel parameters using weighted maximum likelihood as discussed before. Algorithm 1 shows how the different steps of our approach fit together.

Algorithm 1 REPS with RKHS embeddings

```
for $i = 1, \dots, \mathrm{max\_iteration}$ do
    generate rollouts according to $\tilde{\pi}_{i-1}$
    minimize kernel-based dual:
        $\eta^*, \boldsymbol{\alpha}^* \leftarrow \arg\min g(\eta, \boldsymbol{\alpha})$                Eq. 10
    calculate kernel embedding strengths:
        $\boldsymbol{\beta}_j \leftarrow (\mathbf{K}_{\mathrm{sa}} + \lambda \mathbf{I})^{-1} \mathbf{k}_{\mathrm{sa}}(\mathbf{s}_j, \mathbf{a}_j)$        Sec. 2.1
    calculate kernel-based Bellman errors:
        $\delta_j \leftarrow \mathcal{R}_j + \boldsymbol{\alpha}^{*T} \left( \tilde{\mathbf{K}}_{\mathrm{s}} \boldsymbol{\beta}_j - \mathbf{k}_{\mathrm{s}}(\mathbf{s}_j) \right)$        Eq. 11
    calculate the sample weights:
        $w_j \leftarrow \exp(\delta_j / \eta^*)$                Sec. 2.3
    fit a generalizing non-parametric policy:
        $\tilde{\pi}_i(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mu(\mathbf{s}; \mathbf{w}), \sigma^2(\mathbf{s}; \mathbf{w}))$        Sec. 2.3
end for
```

2.4 Hyperparameter Optimization

The computation of the conditional operator has open hyperparameters, i.e., the hyperparameters of the kernels over $\mathbf{s}$ and $\mathbf{a}$ as well as the regularization parameter λ. We set λ and the hyperparameters of kernel k_{a} through minimizing the cross-validation objective $\sum_{i=1}^{n} \left\| \boldsymbol{\phi}(\tilde{\mathbf{s}}_i)^T \boldsymbol{\phi}(\mathbf{s}'_i) - \boldsymbol{\phi}(\tilde{\mathbf{s}}_i)^T C \boldsymbol{\psi}(\mathbf{s}_i, \mathbf{a}_i) \right\|^2$, which minimizes the difference between actual and predicted embedding strengths. This objective is based on the cross-validation objective proposed by [Grünewälder et al., 2012a], but exploits the fact that the embedding will only be evaluated at known functions $\boldsymbol{\phi}(\tilde{\mathbf{s}})$.

The hyperparameters of k_{s}, the kernel for the predicted variable $\mathbf{s}'$, cannot be tuned this way, since trivial solutions exist with essentially constant $\boldsymbol{\phi}(\mathbf{s}')$ by setting the bandwidth is very high. Instead, we set the hyperparameters of k_{s} through minimization of the mean squared Bellman error in a cross-validation procedure. We choose this objective since minimizing the (residual) Bellman error is a common objective for feature selection [Parr et al., 2008] in reinforcement learning.

For the Gaussian process policy, we optimize the kernel hyperparameters separately. The optimization objective is the weighted marginal likelihood, with weights w_i as discussed in Sec. 2.3. This objective is maximized in a two-fold cross-validation procedure.

3 Related Work

RKHS embeddings have been used to model transition dynamics in observable [Grünewälder et al., 2012b] and partially observable [Boots et al., 2013, Nishiyama et al., 2012] domains. Subsequently, the learned transitions models have been used in reinforcement learning methods [Grünewälder et al., 2012b, Nishiyama et al., 2012]. These methods consider only discrete action sets and use greedy maximization with respect to approximated value functions. Consequentially, these methods do not attempt to find a policy that *explores* the MDP in the next iteration.

Rawlik et al. [2013] presents a method that considers continuous-time systems with continuous actions. This method assumes the environments injects observable control noise and that the system is control-affine. Another non-parametric method, proposed by Pazis and Parr [2011], assumes the value function is Lipschitz. This method is model free and only works for deterministic dynamics.

The actions chosen by these methods are deterministic,

[5]The optimization problem and its reduction to a maximum likelihood estimate are given in Appendix 3.

such that if exploration of the state space is required heuristics such as ϵ-greedy or a softmax policy should be used (in the work of Rawlik et al. [2013], a stochastic environment provides explorative excitation). In [Grünewälder et al., 2012b, Nishiyama et al., 2012] it is assumed that the state-action space can be sampled uniformly, which is unrealistic for real systems.

Policy-search methods can be applied to address these shortcomings, by iteratively improving a policy. Bagnell and Schneider [2003] developed a policy gradient method using RKHS embedding of a desirability function that defines a policy. However, their approach is restricted to discrete actions, and as a model-free method, it cannot exploit learnable system dynamics

Deisenroth and Rasmussen [2011] describe a model-based iterative method. They explicitly marginalize the uncertain model to avoid over-greedy optimization. However, their method requires the reward function to be known and to be of squared exponential form, and does not address the exploration problem.

REPS is an alternative that has been shown to work well on a variety of problems [Lioutikov et al., 2014, Kupcsik et al., 2013, Peters et al., 2010, Daniel et al., 2013]. These approaches assume the Lagrangian multiplier V is linear in manually-defined features. We generalize this assumption by requiring V to be a member of any RKHS, allowing implicit infinite feature representations. In contrast to these approaches, our method can naturally handle non-parametric policies. Another alternative to manual feature design for high-dimensional states is to learn features based on a reconstruction objective, as was demonstrated by [Mattner et al., 2012]. In that work, the additional learning step made learning slower.

So far, work on learned transition models for REPS has been limited. The transition dynamics have been approximated by deterministic single-sample outcomes [Daniel et al., 2013, Peters et al., 2010] which only works well for deterministic environments, or by time-dependent linear models [Lioutikov et al., 2014]. Gaussian process models have been used in the bandit setting [Kupcsik et al., 2013] to learn a simulator that predicts the outcome of new rollouts.

In our experiments, we compare the proposed method to model-free and feature-based variants of REPS. Furthermore, we compare our method to the non-parametric value-function methods proposed by Grünewälder et al. [2012b] and Pazis and Parr [2011].

4 Experiments

We evaluate our contribution on a reaching task, as well as two variations of the underpowered pendulum swing-up task. First, we consider a standard version of the task where the agent has access to the angle θ and angular velocity $\dot{\theta}$ directly. In a second version the robot has access only to rendered images, resulting in a high-dimensional input space. In this section we will first discuss elements of our set-up that are the same across tasks, and then discuss implementation specifics and results for each tasks separately.

4.1 Experiment Set-up

We do not consider it possible for our agent to explore by choosing arbitrary state-action pairs. Instead, as shown in Alg. 1, from an initial state distribution our agent explores using its stochastic policy. After every 10 rollouts, the model learner and the policy of the agents are updated. To bootstrap the model and the policy, the agent is given 30 rollouts using a random exploratory policy initially. To avoid excessive computations, we include a simple forgetting mechanism that only keeps the latest 30 rollouts at any time[6]. As each roll-out contains 49 samples on expectation in the larger tasks, most computations are performed on approximately 1500×1500 matrices.

After each update, the learning progress is evaluated by running the learned policy on 100 rollouts with a fixed random seed. This data is not used for learning.

In our experiments, for every method, we performed 10 trials, each consisting of 20 iterations so that 220 rollouts were performed per trial (30 initial rollouts plus 10 per iteration). The model and policy are refined incrementally in every iteration.

4.2 Comparative Methods

We compared multiple methods to learn the tasks. On the one hand, we consider the non-parametric value-function based methods introduced by Grünewälder et al. [2012b] and Pazis and Parr [2011]. We also compare to versions of REPS that use the sample-based model approximation introduced by Daniel et al. [2013], and one that uses a fixed feature set.

Sample based model. Since REPS only needs $\mathbb{E}_{\mathbf{s}'}[f(\mathbf{s}')|\mathbf{s},a]$ at observed state-action pairs $(\mathbf{s}_i, a_i)$, if the system is deterministic this expectation is simply $f(\mathbf{s}'_i)$ at the observed values for $\mathbf{s}_i$. In other systems, this sample-based method is used as approximation.

Feature based REPS Instead of the non-parametric form of V assumed in this paper, we can follow earlier work and define a fixed feature basis [Pe-

[6]This means the q distribution is a mixture of the previous three state-action distributions in our experiments.

ters et al., 2010, Daniel et al., 2013]. We choose to use a similar number of the same radial basis functions used in the non-parametric method, but distributed according to a grid over the state-action space.

Approximate value iteration. In this approach by Grünewälder et al. [2012b], the value function is assumed to be an element of the chosen RKHS. The maximization of the Q function requires discretizing a, for which we choose 25 uniform bins in the allowable range. A deterministic policy selects $a^* = \arg\max Q(\mathbf{s}_i, \cdot)$, but this policy does not explore. To obtain an exploration-exploitation trade-off we replace the maximum by the soft-max operator $a^* \propto \exp(c.Q(\mathbf{s}_i, \cdot)/\mathrm{stdev}(Q(\mathbf{s}_i, \cdot)))$. The free parameter c specifies the greediness of the exploration/exploitation trade-off. In an **on-policy** scheme, the new policy is used to obtain samples for the next iteration.

As a comparison to this on-policy scheme, we also compare using a **grid** of state-action pairs as training data. For a dense grid, this method has a richer input than all other methods, as they start with uninformed roll-outs from the initial-state distribution.

Non-parametric approximate linear programming. Pazis and Parr [2011] introduce a non-parametric method, NPALP, that assumes the value function is Lipschitz. This allows the RL problem to be formalized as a linear program. A greedy policy is obtained that is optimal if all state-action pairs have been visited. Since this is infeasible in practice, we add exploration by adding Gaussian distributed noise to the action in a fraction ϵ of selected actions.

4.3 Reaching Task Experiment

In the reaching task, we simulate a simple two-link planar robot. We assume a perfect inverse dynamics model, so the actions directly set accelerations of the two joints. Each link is of unit length and mass, and the system is completely deterministic. The robot gets negative reinforcement according to the square of the applied action and of the distance of its end-effector to the Cartesian position $\mathbf{x}_{\mathrm{des}} = [0.5, 0]$: $r(\mathbf{s}, \mathbf{a}) = 10^{-4}||\mathbf{a}||_2^2 + ||\mathbf{x} - \mathbf{x}_{\mathrm{des}}||_2^2$. Note that actions are two dimensional and states are four dimensional (joint positions and velocities). The robot starts stretched-out with the end-effector at [0, 2]. The maximum applied acceleration is $50\mathrm{ms}^2$. We use $\gamma = 0.96$.

We use the commonly used exponential-squared (or Gaussian) kernel for angular velocities $\dot{\boldsymbol{\theta}}$ and actions. This kernel is defined as $k_{\mathrm{es}}(\mathbf{x}_i, \mathbf{x}_j) = \exp(-(\mathbf{x}_i - \mathbf{x}_j)^T D(\mathbf{x}_i - \mathbf{x}_j))$ (with D a diagonal matrix containing free parameters). However, for the angles $\boldsymbol{\theta}$ we need a kernel that

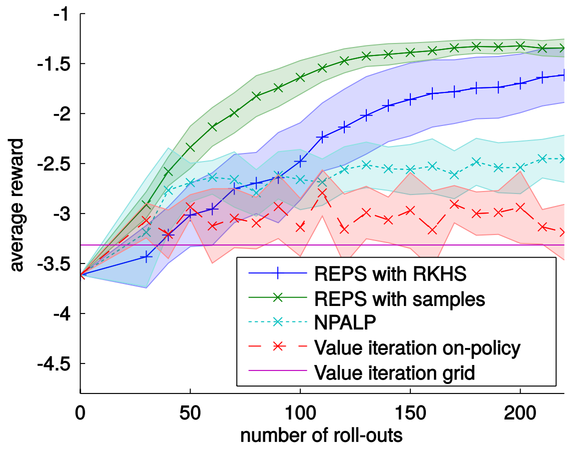

Figure 1: Learning progress of the different methods. Our relative entropy method using the RKHS-embeddings outperforms the other learners. Error bars show twice the standard error of the mean. The value-iteration method does not depend on roll-outs, its performance is shown for comparison.

represents its periodicity. We chose the kernel $k_{\mathrm{p}}(x_i, x_j) = \exp(-\sum_d \sin((x_i^{(d)} - x_j^{(d)})/(2\pi))^2/[\mathbf{l}]_d^2)$, with $\mathbf{l}$ free parameters. Consequently, our complete kernel $k((\boldsymbol{\theta}_i, \dot{\boldsymbol{\theta}}_i, a_i), (\boldsymbol{\theta}_j, \dot{\boldsymbol{\theta}}_j, a_j)) = k_{\mathrm{p}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) k_{\mathrm{se}}(\dot{\boldsymbol{\theta}}_i, \dot{\boldsymbol{\theta}}_j) k_{\mathrm{se}}(a_i, a_j)$.

Feature-based REPS needs a grid over the complete state-action space. Because of practical difficulties with the six-dimensional state-action space, we omitted this method. The exploration parameter ϵ of the NPALP method was set to 0.1, with the standard deviation of Guassian noise added set to $30Nm^2$. The Lipschitz constant was set to 1 with the velocity dimensions scaled by 1/5 for calculating distances. The exploration parameter c of the value iteration method was set to 1.5. These values were manually tuned. For REPS, we use a KL-bound ϵ of 0.5 in our experiments.

Results of the reaching task The results of the reaching task are shown in Figure 1. As this task is deterministic, the sample-based model is optimal and provides a upper bound to the performance we can expect to get. Since REPS with RKHS embeddings needs to iteratively learn the model, its convergence is slower. After inspection of individual trials, the wide variance seems to be caused by occasional failures to find good hyperparameters for the state-action kernel.

The baseline methods NPALP and value iteration using RKHS embeddings obtain good performance after the couple of iterations, but are not suitable for iterative learning as they stop improving after that. The grid based value iteration scheme fails in this case: due

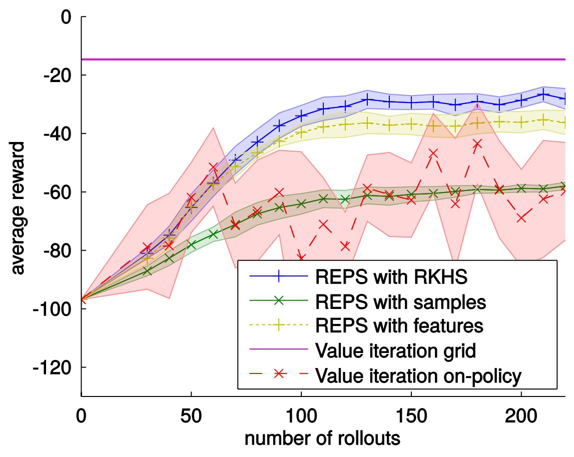

Figure 2: Learning progress of the different methods. Our relative entropy method using the RKHS-embedding of the transition model outperforms the other model learners we considered. Error bars show twice the standard error of the mean.

to memory limitations the maximum grid size we could use was $[5 \times 5 \times 5 \times 5 \times 2 \times 2]$ in the 6-dimensional space, which seems to be insufficient to learn the task.

4.4 Low-Dimensional Swing-up Experiment

In this experiment, we simulate a pendulum with a length of $l = 0.5$m and a mass $m = 10$kg distributed along its length. A torque a can be applied at the pivot. The pendulum is modeled by the dynamics equation $\ddot{\theta} = (glm \sin\theta + a - k\dot{\theta})/(ml^2/3)$, where $k = 0.25$Ns is a friction coefficient and $g = 9.81$ is the gravitational constant. The control frequency is 20 Hz: every 0.05s the agent gets a reward and chooses a new action. The maximum torque that can be applied is 30Nm, which prevents a direct swing-up from the downwards position. Additive noise is applied to the controls with a variance of $1/dt$, resulting in a standard deviation of about 4.5Nm per time step. The reward function we use is $r(s,a) = -10\theta^2 - 0.1\dot{\theta}^2 - 10^{-3}a^2$, where θ is mapped to $[-0.5\pi, 1.5\pi)$ to differentiate the rewards of clockwise and counterclockwise swing-ups. We use a reset probability of 0.02 ($\gamma = 0.98$).

The algorithms are given the angle θ and the angular velocity $\dot{\theta}$ directly, so that the state $\mathbf{s} = [\theta, \dot{\theta}]^T$. The kernels used are the same as in the reaching task experiment (Sec. 4.3). The NPALP method was not designed for stochastic systems and is consequently omitted. We set the grid size for feature-based REPS to $[10 \times 10 \times 10]$ and for the value-iteration method to $[19 \times 11 \times 11]$. The greediness parameter c for on-policy value-iteration was set to 2 after manual tuning.

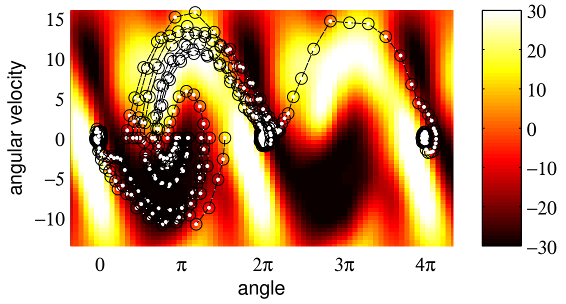

Figure 3: Mean of the learned stochastic policy. Overlayed are 15 trajectories starting at the x-axis between 0 and 2π. Most rollouts reach the desired inverted pose, possibly after one swing back and forth. One rollout overshoots and makes a full rotation before stabilizing the pendulum.

Results of the low-dimension swing-up. A comparison between the methods we implemented is shown in Fig. 2. The value iteration methods starts out competitively, but fails to keep improving the policy. Large variance indicates the learning process is unstable. The bounded policy update in REPS makes learning progress smooth by limiting information loss, and trades off exploration and exploitation.

The sample-based model learner performs considerably worse in this experiment, as it cannot account for stochastic transitions. The variant with fixed features performs well initially, but in later iterations the non-parametric method focuses its representative power on frequently visited parts of the state-space, thus performing better.

The grid-based value iteration method works well. The policy learned by our method, shown in Fig. 3, sometimes overshoots the inverted position, which the grid based value iteration avoids. However, providing the grid is only possible in simulation, as without an existing controller it is generally not possible to start the dynamical system with arbitrary position and velocity.

4.5 High-Dimensional Swing-up Experiment

In a more challenging version of the swing-up, the agent only has access to images of the pendulum. We render an image of the pendulum in its current state, as well as a difference image between visual representations of the pendulum in its current- and previous state (to provide a notion of angular velocity).

The rendered images are blurred with a Gaussian filter with a standard deviation of 20% of the image width to enhance the generalizability. Subsequently, both images are resized to 20×20 pixels, to obtain a 800 dimensional representation, as illustrated in Fig. 4.

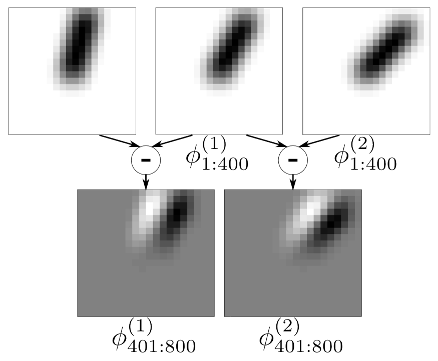

Figure 4: Illustration of high-dimensional features. First row: Low-resolution images of the pendulum are rendered to yield the first 400 dimensions of the feature vector. Second row: the difference images form the next 400 dimensions of the feature vector.

For many reinforcement learning algorithms such high-dimensional continuous state-spaces would be infeasible, as using generic features [Lagoudakis and Parr, 2003] yields impracticable feature dimensions on 800-dimensional observations. For example, there are 800^2 second-degree polynomials, or 2^{800} radial basis functions on the smallest possible grid. All comparative methods that performed reasonably rely on a grid, so we will compare our method only to our solution from the previous experiment.

We choose squared-exponential kernels for all dimensions. To reduce the number of hyperparameters, we constrain the bandwidth on all pixels per image to be the same. So, the number of automatically tuned model parameters is 4 (including one for the kernel on actions, which is again a squared exponential).

Results for the high-dimensional swing-up. The results of the evaluation of our approach, together with the sample-based baseline method, are shown in Fig. 5. We see that the learning progress is a bit slower than progress on the low-dimensional task. However, the final solution is equally good.

Generally, given a kernel that yields appropriate similarity values, many different representations could be used: in the end, the algorithm only performs operations based on those simililarity values. Some pixels will never change their value. This is not problematic, as stationary kernels such as the squared exponential are not influenced by these features.

5 Discussion and Future Work

In this paper, we have developed a policy search method with smooth, robust updates to solve continuous MDPs. This method uses learned non-parametric

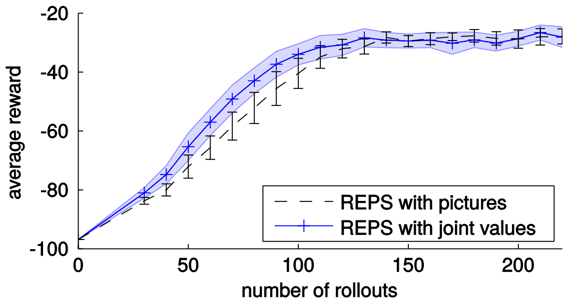

Figure 5: Learning progress of our method on the underpowered pendulum swing-up directly from joint values and from rendered images. Shaded area is twice the standard error of the mean.

models and allows the use of non-parametric policies, avoiding hand-crafted features. By embedding the conditional transition distribution, expectations over functions of the next state can be calculated without density estimation. The resulting predictions are robust even in high-dimensional state spaces.

Many tasks concerning sensory data hava a high extrinsic dimensionality, but are intrinsically low-dimensional. Kernel-based algorithms perform all operations on kernel values, so they are invariant to the extrinsic dimensionality. Our kernel-based RL algorithm can be thus be applied to such tasks without the separate dimension reduction step used in [Mattner et al., 2012].

We show the resulting algorithm to be able to outperform other algorithms on a reaching task and a pendulum swing-up task with control noise. Our algorithm does this using only on-policy samples, i.e., without the need to sample arbitrary state-action pairs. We also show the algorithm is able to solve the task using only rendered images, with an 800-dimensional representation of the state space.

In future work, we want to investigate long-term planning using RKHS embeddings and investigate synergies between the optimization problem and the generalizing policy. We will also adress hyperparameter optimization in systems with multi-dimensional controls, that currently does not always yield desired results. We plan to apply our algorithm on real-world robotic tasks, and investigate how to exploit structural knowledge about dynamical systems in such tasks.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreements #610967 (TACMAN) and #270327 (CompLACS).

References

J.A. Bagnell and J. Schneider. Policy search in reproducing kernel Hilbert space. Technical Report RI-TR-03-45, CMU, 2003.

L. Baird and A. W. Moore. Gradient descent for general reinforcement learning. *Advances in Neural Information Processing Systems*, pages 968–974, 1999.

B. Boots, A. Gretton, and G. J. Gordon. Hilbert space embeddings of predictive state representations. In *International Conference on Uncertainty in Artificial Intelligence*, 2013.

C. Daniel, G. Neumann, O. Kroemer, and J. Peters. Learning sequential motor tasks. In *International Conference on Robotics and Automation*, 2013.

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*, pages 465–472, 2011.

S. Grünewälder, G. Lever, L. Baldassarre, S. Patterson, A. Gretton, and M. Pontil. Conditional mean embeddings as regressors. In *International Conference on Machine Learning*, pages 1823–1830, 2012a.

S. Grünewälder, G. Lever, L. Baldassarre, M. Pontil, and A. Gretton. Modelling transition dynamics in MDPs with RKHS embeddings. In *International Conference on Machine Learning*, pages 535–542, 2012b.

J. Kober, E. Oztop, and J. Peters. Reinforcement learning to adjust robot movements to new situations. In *International Joint Conference on Artificial Intelligence*, pages 2650–2655, 2011.

A.G. Kupcsik, M.P. Deisenroth, J. Peters, and G. Neumann. Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the National Conference on Artificial Intelligence*, 2013.

M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4: 1107–1149, December 2003.

R. Lioutikov, A. Paraschos, G. Neumann, and J. Peters. Sample-based information-theoretic stochastic optimal control. In *International Conference on Robotics and Automation*, 2014.

J. Mattner, S. Lange, and M. Riedmiller. Learn to swing up and balance a real pole based on raw visual input data. In *International Conference on Neural Information Processing*, pages 126–133, 2012.

Y. Nishiyama, A. Boularias, A. Gretton, and K. Fukumizu. Hilbert space embeddings of POMDPs. In *International Conference on Uncertainty in Artificial Intelligence*, pages 644–653, 2012.

R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *International Conference on Machine learning*, pages 752–759, 2008.

J. Pazis and R. Parr. Non-parametric approximate linear programming for MDPs. In *AAAI Conference on Artificial Intelligence*, pages 459–464, 2011.

J. Peters, K. Muelling, and Y. Altun. Relative entropy policy search. In *National Conference on Artificial Intelligence (AAAI), Physically Grounded AI Track*, pages 1607–1612, 2010.

K. Rawlik, M. Toussaint, and S. Vijayakumar. Path integral control by reproducing kernel Hilbert space embedding. In *International Joint Conference on Artificial Intelligence*, 2013.

B. Schölkopf, R. Herbrich, and A.J. Smola. A generalized representer theorem. In David Helmbold and Bob Williamson, editors, *Computational Learning Theory*, volume 2111 of *Lecture Notes in Computer Science*, pages 416–426. Springer Berlin Heidelberg, 2001.

L. Song, K. Fukumizu, and A. Gretton. Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models. *Signal Processing Magazine, IEEE*, 30(4): 98–111, 2013.

A. Zimin and G. Neu. Online learning in episodic Markovian decision processes by relative entropy policy search. In *Advances in Neural Information Processing Systems*, pages 1583–1591, 2013.

Appendix

Herke van Hoof[‡] **Jan Peters**[‡*] **Gerhard Neumann**[‡]
[‡]TU Darmstadt, Computer Science Department. *MPI for Intelligent Systems.

1 Derivation of REPS Solution

We start out with the constrained optimization problem

$$\max_{\pi,\mu_\pi} J(\pi) = \max_{\pi,\mu_\pi} \iint_{A\times S} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{R}_{\mathbf{s}}^{\mathbf{a}} \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} \quad (1)$$

$$\begin{aligned} s.t. \quad \iint_{A\times S} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} &= 1 \quad (2)\\ \forall s' \quad \iint_{A\times S} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} &= \mu_\pi(\mathbf{s}') \quad (3)\\ \iint_{A\times S} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s}) \log \frac{\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})}{q(\mathbf{s},\mathbf{a})}\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} &\leq \epsilon. \quad (4) \end{aligned}$$

For every constraint, we introduce a Lagrangian multiplier. Because (3) represents a continuum of constraints, we integrate over the value of this constraint multiplied by a state-dependent Lagrangian multiplier $V(\mathbf{s})$. We will write $p(\mathbf{s},\mathbf{a}) = \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})$ to keep the exposition brief. Therefore, the Lagrangian

$$\begin{aligned} L(p,\eta,V,\lambda) = &\iint_{A\times S} p(\mathbf{s},\mathbf{a})\mathcal{R}_{\mathbf{s}}^{\mathbf{a}}\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}\\ &+ \lambda\left(1 - \iint_{A\times S} p(\mathbf{s},\mathbf{a})\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}\right)\\ &+ \int_S V(\mathbf{s}')\left(\iint_{A\times S} p(\mathbf{s},\mathbf{a})\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} - \mu_\pi(\mathbf{s}')\right)\mathrm{d}\mathbf{s}'\\ &+ \eta\left(\epsilon - \iint_{A\times S} p(\mathbf{s},\mathbf{a}) \log\frac{p(\mathbf{s},\mathbf{a})}{q(\mathbf{s},\mathbf{a})}\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}\right). \end{aligned}$$

The Lagrangian can be re-shaped, using $\mu_\pi(\mathbf{s}) = \int_A p(\mathbf{s},\mathbf{a})\mathrm{d}\mathbf{a}$, in the more convenient form

$$\begin{aligned} L(p,\eta,V,\lambda) &= \lambda - \mathbb{E}_{p(\mathbf{s},\mathbf{a})}\left[V(\mathbf{s})\right] + \eta\epsilon\\ &+ \mathbb{E}_{p(\mathbf{s},\mathbf{a})}\left[\mathcal{R}_{\mathbf{s}}^{\mathbf{a}} - \lambda + \int_S V(\mathbf{s}')\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}\mathrm{d}\mathbf{s}' - \eta\log\frac{p(\mathbf{s},\mathbf{a})}{q(\mathbf{s},\mathbf{a})}\right]. \end{aligned}$$

To find the optimal p, we take the derivative of L w.r.t. p and set it to zero

$$\begin{aligned} 0 =& \frac{\partial L}{\partial p(\mathbf{s},\mathbf{a})}\\ =& \mathcal{R}_{\mathbf{s}}^{\mathbf{a}} - \lambda + \int_S V(\mathbf{s}')\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}\mathrm{d}\mathbf{s}' - \eta\log\frac{p(\mathbf{s},\mathbf{a})}{q(\mathbf{s},\mathbf{a})} - \eta - V(\mathbf{s}) \end{aligned}$$

therefore,

$$\eta\log\frac{p(\mathbf{s},\mathbf{a})}{q(\mathbf{s},\mathbf{a})} = \mathcal{R}_{\mathbf{s}}^{\mathbf{a}} - \lambda + \int_S V(\mathbf{s}')\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}\mathrm{d}\mathbf{s}' - \eta - V(\mathbf{s})$$

$$\begin{aligned} p(\mathbf{s},\mathbf{a}) =& q(\mathbf{s},\mathbf{a})\exp\left(\frac{\mathcal{R}_{\mathbf{s}}^{\mathbf{a}} - \int_S V(\mathbf{s}')\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}\mathrm{d}\mathbf{s}' - V(\mathbf{s})}{\eta}\right)\\ &\cdot\exp\left(\frac{-\lambda-\eta}{\eta}\right)\\ \propto& q(\mathbf{s},\mathbf{a})\exp\left(\frac{\mathcal{R}_{\mathbf{s}}^{\mathbf{a}} - \int_S V(\mathbf{s}')\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}\mathrm{d}\mathbf{s}' - V(\mathbf{s})}{\eta}\right). \end{aligned}$$

The function $V(\mathbf{s})$ resembles a value function, so that $\delta(\mathbf{s},\mathbf{a},V) = \mathcal{R}_{\mathbf{s}}^{\mathbf{a}} - \int_S V(\mathbf{s}')\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}}\mathrm{d}\mathbf{s}' - V(\mathbf{s})$ can be identified as a Bellman error. Since $p(\mathbf{s},\mathbf{a})$ is a probability distribution we can identify $\exp(-\lambda/\eta - 1)$ to be a normalization factor

$$\begin{aligned} Z^{-1} &= \left(\iint_{A\times S} q(\mathbf{s},\mathbf{a})\exp\left(\delta(\mathbf{s},\mathbf{a},V)/\eta\right)\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}\right)^{-1}\\ &= \left(\mathbb{E}_{q(\mathbf{s},\mathbf{a})}\exp\left(\delta(\mathbf{s},\mathbf{a},V)/\eta\right)\right)^{-1}. \end{aligned}$$

2 The Dual and its Derivatives

We can re-insert the state-action probabilities in the Lagrangian to obtain the dual

$$\begin{aligned}g(\eta, V, \lambda) =&\lambda + \eta\epsilon\\ &+ \mathbb{E}_{p(\mathbf{s},\mathbf{a})}\left[\delta(\mathbf{s},\mathbf{a},V) - \lambda - \eta\log\frac{p(\mathbf{s},\mathbf{a})}{q(\mathbf{s},\mathbf{a})}\right]\\ =&\lambda + \eta\epsilon + \mathbb{E}_{p(\mathbf{s},\mathbf{a})}\left[-\lambda+\lambda\right]\\ &+ \mathbb{E}_{p(\mathbf{s},\mathbf{a})}\left[\delta(\mathbf{s},\mathbf{a},V) - \delta(\mathbf{s},\mathbf{a},V) + \eta\right]\\ =&\lambda + \eta\epsilon + \mathbb{E}_{p(\mathbf{s},\mathbf{a})}\eta \; \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}\\ =&\lambda + \eta\epsilon + \eta = \eta\epsilon + \eta\log(Z)\\ =&\eta\epsilon + \eta\log\left(\mathbb{E}_{q(\mathbf{s},\mathbf{a})}\exp\left(\delta(\mathbf{s},\mathbf{a},V)/\eta\right)\right),\end{aligned}$$

where we used the identity

$$\begin{aligned}\exp\left(-\lambda/\eta - 1\right) &= Z^{-1}\\ \lambda + \eta &= \eta\log(Z).\end{aligned}$$

The expected value over q can straightforwardly be approximated by taking the average of samples $1, \ldots, n$ taken from q. Note that λ and q do not appear in the final expression.

$$g(\eta, V) = \eta\epsilon + \eta\log\left(\frac{1}{n}\sum_{i=1}^{n}\exp\left(\delta(\mathbf{s}_i,\mathbf{a}_i,V)/\eta\right)\right).$$

When employing the kernel embedding, the Bellman error is written as

$$\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha}) = \mathcal{R}_{\mathbf{s}_i}^{\mathbf{a}_i} + \boldsymbol{\alpha}^T(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_\mathrm{s}(\mathbf{s}_i)).$$

We define

$$w_i = \frac{\exp\left(\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})/\eta\right)}{\sum_{i=j}^{n}\exp\left(\delta(\mathbf{s}_j,\mathbf{a}_j,\boldsymbol{\alpha})/\eta\right)}$$

to keep equations brief and readable. The partial derivatives can be written as:

$$\begin{aligned}\frac{\partial g(\eta,\boldsymbol{\alpha})}{\partial\eta} =& -\frac{1}{\eta}\sum_{i=1}^{n}w_i\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha}) + \epsilon\\ &+ \log\left(\frac{1}{n}\sum_{i=1}^{n}\exp\left(\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})/\eta\right)\right),\end{aligned}$$

$$\frac{\partial g(\eta,\boldsymbol{\alpha})}{\partial\boldsymbol{\alpha}} = \sum_{i=1}^{n}w_i\left(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_\mathrm{s}(\mathbf{s}_i)\right),$$

and furthermore, for the Hessian we obtain

$$\begin{aligned}\frac{\partial^2 g(\eta,\boldsymbol{\alpha})}{\partial\eta\partial\eta} =& \frac{1}{\eta}\sum_{i=1}^{n}w_i\left(\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})\right)^2\\ &- \frac{1}{\eta}\left(\sum_{i=1}^{n}w_i\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})\right)^2\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 g(\eta,\boldsymbol{\alpha})}{\partial\boldsymbol{\alpha}\partial\boldsymbol{\alpha}^T} =& -\frac{1}{\eta}\sum_{i=1}^{n}w_i\left(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_\mathrm{s}(\mathbf{s}_i)\right)\\ &\cdot\sum_{i=1}^{n}w_i\left(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_\mathrm{s}(\mathbf{s}_i)\right)^T +\\ &\sum_{i=1}^{n}\frac{w_i}{\eta}\left(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_\mathrm{s}(\mathbf{s}_i)\right)\left(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_\mathrm{s}(\mathbf{s}_i)\right)^T,\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 g(\eta,\boldsymbol{\alpha})}{\partial\eta\partial\boldsymbol{\alpha}} =& -\frac{1}{\eta}\sum_{i=1}^{n}w_i\left(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_\mathrm{s}(\mathbf{s}_i)\right)\\ &+ \sum_{i=1}^{n}\frac{w_i}{\eta}\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})\sum_{i=1}^{n}w_i\left(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_\mathrm{s}(\mathbf{s}_i)\right)\\ &+ \frac{1}{\eta}\sum_{i=1}^{n}w_i\left(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_\mathrm{s}(\mathbf{s}_i)\right)\\ &- \frac{1}{\eta}\sum_{i=1}^{n}w_i\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})\left(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_\mathrm{s}(\mathbf{s}_i)\right)\\ &.\end{aligned}$$

3 Fitting a Generalizing Policy to State-Action Samples

To fit a generalizing policy $\tilde{\pi}(\mathbf{a}|\mathbf{s};\boldsymbol{\theta})$ to the samples-based policy $p(\mathbf{s}_i,\mathbf{a}_i) = \pi(\mathbf{a}_i|\mathbf{s}_i)\mu_\pi(\mathbf{s}_i)$ (defined only on samples $i \in \{1,\ldots,n\}$), we minize the expected Kullback-Leibler divergence

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mu_\pi(\mathbf{s})} \mathrm{KL}(\pi(\mathbf{a}|\mathbf{s})||\tilde{\pi}(\mathbf{a}|\mathbf{s}))\\ &= \int_S \mu_\pi(\mathbf{s}) \int_A \pi(\mathbf{a}|\mathbf{s}) \log \frac{\pi(\mathbf{a}|\mathbf{s})}{\tilde{\pi}(\mathbf{a}|\mathbf{s};\boldsymbol{\theta})} \mathbf{d}\mathbf{a}\mathbf{d}\mathbf{s}.\end{aligned}$$

This is a standard objective for matching two distributions. Note that the alternative Kullback-Leibler divergence $\mathrm{KL}(\tilde{\pi}(\mathbf{a}|\mathbf{s})||\pi(\mathbf{a}|\mathbf{s}))$ is undefined since $\pi(\mathbf{a}|\mathbf{s})$ is 0 at most places. Since the contribution to the integral is 0 for any $(\mathbf{s},\mathbf{a}) \notin \{(\mathbf{s}_1,\mathbf{a}_1),\ldots,(\mathbf{s}_n,\mathbf{a}_n)\}$, we can equivalently write:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^n \mu_\pi(\mathbf{s}_i)\pi(\mathbf{a}_i|\mathbf{s}_i) \log \frac{\pi(\mathbf{a_i}|\mathbf{s}_i)}{\tilde{\pi}(\mathbf{a}_i|\mathbf{s}_i;\boldsymbol{\theta})}\\ &= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^n \mu_\pi(\mathbf{s}_i)\pi(\mathbf{a}_i|\mathbf{s}_i) \log \frac{1}{\tilde{\pi}(\mathbf{a}_i|\mathbf{s}_i;\boldsymbol{\theta})}\\ &\quad + \sum_{i=1}^n \mu(\mathbf{s}_i)\pi(\mathbf{a}_i|\mathbf{s}_i)\log(\pi(\mathbf{a}_i|\mathbf{s}_i))\\ &= \arg\max_{\theta} \sum_{i=1}^n \mu_\pi(\mathbf{s}_i)\pi(\mathbf{a}_i|\mathbf{s}_i)\log\tilde{\pi}(\mathbf{a}_i|\mathbf{s}_i;\boldsymbol{\theta})\end{aligned}$$

where we used the fact that we can subtract terms constant in $\boldsymbol{\theta}$ and apply monotonously increasing functions to the terms to be minimized without changing the location of the minimum. Note that the final result is simply a weighted maximum-likelyhood estimate of $\boldsymbol{\theta}$. This result can be used to fit a parametric policy, or, as we demonstrate in the main material, a non-parametric policy to the weighted samples.

4 Optimization with Respect to V

In order to show that we can minimize the dual function g, we need to show that the optimal solution of the value function has the following form

$$V^* = \sum_{\tilde{\mathbf{s}}\in\tilde{\mathcal{S}}} \alpha_{\tilde{\mathbf{s}}} k_{\mathrm{s}}(\tilde{\mathbf{s}},\cdot) \tag{5}$$

We follow some steps in the proof of Schölkopf et al. [2001]. They consider arbitrary functions c mapping to $\mathbb{R}\cup\{\infty\}$ of the form

$$c((\mathbf{s}_1,y_1,V(\mathbf{s}_1)),\ldots,(\mathbf{s}_m,y_m,V(\mathbf{s}_m))), \tag{6}$$

which typically defines an error function of function $V(\mathbf{s})$ on the samples $\mathbf{s}_i$ with desired output y_i. In our case, we do not have desired output values y_i for our objective function. This is inconsequential as c can be arbitrary, and so can be independent of all y values.

Any function V can be written as $V = \sum_{\tilde{\mathbf{s}}\in\tilde{\mathcal{S}}} \alpha_{\tilde{\mathbf{s}}} k_{\mathrm{s}}(\tilde{\mathbf{s}},\cdot) + v(\mathbf{s})$, where $v(\mathbf{s})$ is an additional bias term. If V is constrained to be in the Hilbert space defined by k, Schölkopf et al. [2001] show that c is independent of the bias term $v(\mathbf{s})$. This means that for any optimal V' that is not of the proposed form, there is a V^* of the proposed form that has the same objective value which is obtained by subtracting $v(\mathbf{s})$ from V'.

As the dual function g satisfies the conditions to cost function c, for us this means that there is at least one V^* optimizing g of the proposed form. Note that it is inconsequential that the dual g also depends on Langrangian parameter η. For any optimum (η^*, V'^*), if $V'*$ is not of the proposed form, the projection V^* of V'^* on the proposed basis satisfies $g(\eta^*, V'^*) = g(\eta^*, V^*)$, so (η^*, V^*) must be an optimum as well.