Compressed Conditional Mean Embeddings for Model-Based Reinforcement Learning

Guy Lever
University College London
London, UK
g.lever@cs.ucl.ac.uk

John Shawe-Taylor
University College London
London, UK
j.shawe-taylor@cs.ucl.ac.uk

Ronnie Stafford
University College London
London, UK
r.stafford.12@ucl.ac.uk

Csaba Szepesvári
University of Alberta
Edmonton, Canada
szepesva@cs.ualberta.ca

Abstract

We present a model-based approach to solving Markov decision processes (MDPs) in which the system dynamics are learned using conditional mean embeddings (CMEs). This class of methods comes with strong performance guarantees, and enables planning to be performed in an induced finite (pseudo-)MDP, which approximates the MDP, but can be solved exactly using dynamic programming. Two drawbacks of existing methods exist: firstly, the size of the induced finite (pseudo-)MDP scales quadratically with the amount of data used to learn the model, costing much memory and time when planning with the learned model; secondly, learning the CME itself using powerful kernel least-squares is costly – a second computational bottleneck. We present an algorithm which maintains a rich kernelized CME model class, but solves both problems: firstly we demonstrate that the loss function for the CME model suggests a principled approach to *compressing* the induced (pseudo-)MDP, leading to faster planning, while maintaining guarantees; secondly we propose to learn the CME model itself using fast sparse-greedy kernel regression well-suited to the RL context. We demonstrate superior performance to existing methods in this class of model-based approaches on a range of MDPs.

1 Introduction

Several methods have been proposed for model-based reinforcement learning (RL) which, due to the form of the model, induce a finite (pseudo-)MDP[1] which approximates the MDP, such that solving the induced (pseudo-)MDP delivers a good policy for the true MDP (Ormoneit and Sen 2002; Grünewälder et al. 2012b; Yao et al. 2014). In these methods the learned 'model' can be viewed as a "conditional mean embedding" (CME) of the MDP transition dynamics. We review this family of methods in Section 3.

In general the approach has the following potentially useful properties: *a*) The induced finite approximate Bellman optimality equation can be solved exactly in finite time, avoiding instability associated with approximate dynamic programming (e.g. Bertsekas, 2011). As a result, planning is guaranteed to succeed. *b*) Guarantees on the value of the learned policy exist: e.g. consistency in the large data limit for the algorithm of Ormoneit and Sen, 2002 under mild smoothness assumptions or bounds on the value of the policy learned in terms of the error of the CME (Grünewälder et al. 2012b; Yao et al. 2014). *c*) By reducing model learning to a supervised learning problem, to some extent the problem of a-priori defining a good compact representation architecture for value functions is avoided. *d*) The approach can be data-efficient since state-of-the-art non-parametric approaches can be used to approximate the CME from few data points.

One drawback is the fact that the size of the induced finite (pseudo-)MDP scales quadratically with the amount of data used to learn the model, significantly slowing down the planning step and straining the algorithm's memory requirements as more data is collected. Further, Grünewälder et al., 2012b employ non-parametric methods to learn the CME which are data efficient but computationally expensive. It is therefore of interest to investigate methods for learning CMEs in RL which retain these benefits but enable efficient planning and which can be learned efficiently even for problems that require huge amounts of data to learn a good model. We present a method of learning a *compressed CME* which induces a compressed finite pseudo-MDP for efficient planning, decoupling the amount of data collected and the size of the induced pseudo-MDP. The compression guarantees the existence of compressed CMEs maintaining good performance bounds. We optimize the CME non-parametrically but efficiently, using kernel matching pursuit.

2 Preliminaries

2.1 Reinforcement Learning Background

We recall basic concepts associated with reinforcement learning (RL). In RL an agent acts in an environment by sequentially choosing actions over a sequence of time steps, in order to maximize a cumulative reward. We model this as a *Markov decision process* (MDP) which is defined using a *state space* $\mathcal{S}$, an *action space* $\mathcal{A}$, an *initial state distribution* P_1 over $\mathcal{S}$, an *(immediate) reward function* $r : \mathcal{S} \times \mathcal{A} \to [0,1]$, and a stationary *Markov transition kernel* $(P(s'|s,a))_{(s,a,s')\in\mathcal{S}\times\mathcal{A}\times\mathcal{S}}$. These, together with a (stationary, stochastic, Markov) policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, a map from the state space to the set of distributions over

Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] Pseudo-MDPs relax the potential transition functions from the set of probability measures to signed measures. See Section 3.1.

the action space, gives rise to a controlled Markov chain $\xi = (S_1, A_1, S_2, A_2, \dots)$ where $S_1 \sim P_1$, $A_t \sim \pi(S_t)$ and $S_{t+1} \sim P(\cdot|S_t, A_t)$, i.e., P determines the stochastic dynamics of the controlled process. Generally, we will denote the successor state of s by s'. An *agent* that interacts with an MDP observes (some function) of the states and selects actions sequentially with the goal to accumulate reward comparable to what could be obtained by a policy π^* which maximizes the *expected return* (expected total cumulative discounted reward), $J(\pi) := \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t); \pi]$ where, $\mathbb{E}[\cdot; \pi]$ denotes the expectation with respect to P_1, P and π. We recall the *value function* $V^\pi(s) := \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|S_1 = s; \pi]$ and *action-value function* $Q^\pi(s,a) := r(s,a) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,a)}[V^\pi(S')]$. The optimal value function is defined by $V^*(s) = \sup_{\pi \in \Pi} V^\pi(s)$. For a given action-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ we define the (deterministic) *greedy policy* w.r.t. Q by $\pi(s) := \operatorname{argmax}_{a \in \mathcal{A}} Q(s,a)$ and denote $\pi = \text{greedy}(Q)$ (ties broken arbitrarily). Obtaining V^π for a given π is known as *value estimation*. Any V^π satisfies the *Bellman equation*,

$$V^\pi(s) = \mathbb{E}_{A \sim \pi(s)}[r(s,A) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,A)}[V^\pi(S')]], \quad (1)$$

and the map $T^\pi : \mathcal{V} \to \mathcal{V}$, over the set $\mathcal{V}$ of real-valued functions on $\mathcal{S}$, defined by $(T^\pi V)(s) := \mathbb{E}_{A \sim \pi(s)}[r(s,A) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,A)}[V(S')]]$ is known as the Bellman operator for π. For finite state spaces an optimal policy can be obtained using dynamic programming methods such as value iteration (Bellman 1957) and policy iteration (Howard 1960). In policy iteration V^π is obtained by solving (1) for a given deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ followed by taking the greedy policy with respect to V^π and iterating. In MDPs with large or continuous state spaces, value functions are typically represented in some approximation architecture, e.g. as a linear function in some feature space, $V^\pi(s) \approx \langle v_\pi, \phi(s) \rangle_{\mathcal{F}} =: \hat{V}^\pi(s)$ where $\phi : \mathcal{S} \to \mathcal{F}$ is a feature map, and $\mathcal{F}$ a Hilbert space. Choosing a feature map $\phi(\cdot)$ a-priori can be a problem since it is difficult to balance powerful representation ability and compactness. Further, in the approximate case value estimation entails solving $\langle w_\pi, \phi(s) \rangle_{\mathcal{F}} \approx r(s, \pi(s)) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,\pi(s))}[\langle v_\pi, \phi(S') \rangle_{\mathcal{F}}]$, which must be solved approximately (using, for example, LSTD (Bradtke and Barto 1996) or minimizing the Bellman residual (Baird 1995)) since in general no solution in v_π can be found with equality for a given feature map $\phi(\cdot)$, which can lead to instabilities (Bertsekas 2012).

Model-based reinforcement learning is an approach to RL in which data is used to estimate the dynamics and/or reward function followed by solving the resulting estimated MDP. This approach can be data efficient since the planning stage can be performed offline, and does not require interaction with the system. In this work we will present a model-based policy iteration algorithm with a particular form of the transition model. We suppose that the mean reward function r is known. Trivially, any estimate for the mean reward function can be substituted into our algorithm.

Related Work The approaches most similar to ours are by Ormoneit and Sen, 2002, Grünewälder et al., 2012b and Yao et al., 2014 and we defer a detailed discussion to Section 3.2. Related dynamics models which estimate expected successor feature maps using regression are common in approximate dynamic programming (Parr et al. 2008; Sutton et al. 2008). The key difference is that we will use the model to solve an MDP by deriving an induced finite pseudo-MDP and solving it exactly. Further, we separate the dynamics learning from any particular policy. Similarly van Hoof, Peters, and Neumann, 2015 use a similar transition model in a direct policy search algorithm.

3 Conditional Mean Embeddings for RL

The representation of the model that we study in this work is motivated by dynamic programming algorithms. In policy iteration the model is required to solve the Bellman expectation equation (1), and so it is sufficient to compute the conditional expectation of value functions, $\mathbb{E}_{S' \sim P(\cdot|s,a)}[V(S')]$. Recalling the discussion in Section 2.1, when $V(s) = \langle v, \phi(s) \rangle_{\mathcal{F}}$, we have that $\mathbb{E}_{S' \sim P(\cdot|s,a)}[V(S')] = \langle \mathbb{E}_{S' \sim P(\cdot|s,a)}[\phi(S')], v \rangle_{\mathcal{F}}$ and hence it suffices to learn the function $\mu : \mathcal{S} \times \mathcal{A} \to \mathcal{F}$ such that

$$\mu(s,a) = \mathbb{E}_{S' \sim P(\cdot|s,a)}[\phi(S')]. \quad (2)$$

In this work our "transition model" is an estimate of $\mu(\cdot)$, and for any estimate $\hat{\mu}(\cdot)$ we write,

$$\hat{E}_{\hat{\mu}}(V, (s,a)) = \langle \hat{\mu}(s,a), v \rangle_{\mathcal{F}} \approx \mathbb{E}_{S' \sim P(\cdot|s,a)}[V(S')], \quad (3)$$

where we used the (so-far implicit) convention that $V(s) = \langle v, \phi(s) \rangle_{\mathcal{F}}$. Note that $\hat{E}_{\hat{\mu}}$ may not correspond to any conditional probability measure on $\mathcal{S}$. The loss

$$\text{loss}(\hat{\mu}) := \mathbb{E}_{(s,a) \sim D, s' \sim P(\cdot|s,a)}[||\hat{\mu}(s,a) - \phi(s')||^2_{\mathcal{F}}],$$

where D is some data distribution over $\mathcal{S} \times \mathcal{A}$, serves as a natural objective for the problem of learning $\mu(\cdot)$ (Grünewälder et al. 2012a). Given data[2] $\mathcal{D} = \{(s_i, a_i, s'_i)\}_{i=1}^n$, where $s'_i \sim P(\cdot|s_i, a_i)$ the empirical loss is therefore,

$$\hat{\text{loss}}(\hat{\mu}) := \frac{1}{n} \sum_{i=1}^{n} ||\hat{\mu}(s_i, a_i) - \phi(s'_i)||^2_{\mathcal{F}}. \quad (4)$$

The function $\mu(\cdot)$ defined in (2) is known as the *conditional mean embedding* (CME) of P in $\mathcal{F}$ and learning methods have been provided (Song, Fukumizu, and Gretton 2013; Grünewälder et al. 2012a).

3.1 Key Properties of the CME Approach

The Induced Finite (Pseudo-)MDP As we will see in Section 3.2, several approaches to learn the CME (2) from data result in a solution of the form $\hat{\mu}(s,a) = \sum_{i=1}^n \alpha_i(s,a) \phi(s'_i)$. This means that, from (3),

$$\hat{E}_{\hat{\mu}}(V, (s,a)) = \langle \hat{\mu}(s,a), v \rangle_{\mathcal{F}} = \sum_{i=1}^{n} \alpha_i(s,a) V(s'_i), \quad (5)$$

i.e. our estimates of conditional expectation can be computed by measuring V only on the sample points $\{s'_i\}_{i=1}^n$. In fact $\alpha_i(s,a)$ can be viewed as the "probability"[3] of transitioning to state s'_i from state-action (s,a) under the learned model (and the probability of transitioning to states beyond

[2] We will use the (sloppy) notation $s'_i \in \mathcal{D}$ to indicate the successor states in $\mathcal{D}$ in the sense that $s'_i \in \mathcal{D} \Leftrightarrow \exists (s_i, a_i, s'_i) \in \mathcal{D}$, and similarly for the actions a_i, and predecessor states s_i.

[3] The $\alpha_i(s,a)$ are not necessarily positive or normalized.

the sample is zero). Collect $(\alpha_i(s,a))_{i=1}^n$ into the vector $\boldsymbol{\alpha}(s,a) \in \mathbb{R}^n$. If, further,

$$||\boldsymbol{\alpha}(s'_i, a)||_1 \leq 1 \tag{6}$$

for all successor states $s'_i \in \mathcal{D}$ and $a \in \mathcal{A}$ then the approximation $\hat{T}^\pi_{\hat{\mu}} : \mathcal{V} \to \mathcal{V}$ of the Bellman operator T^π defined by

$$\begin{aligned}(\hat{T}^\pi_{\hat{\mu}} V)(s) &:= \mathbb{E}_{A\sim\pi(s)}[r(s,A) + \gamma \hat{E}_{\hat{\mu}}(V,(s,A))] \\ &= \mathbb{E}_{A\sim\pi(s)}[r(s,A) + \gamma \sum_{i=1}^n \alpha_i(s,A) V(s'_i)] \end{aligned} \tag{7}$$

is a contraction on the sample and so iterated "backups" $V \leftarrow \hat{T}^\pi_{\hat{\mu}} V$ will converge to a fixed point $\hat{V}^\pi_{\hat{\mu}}$ such that $\hat{V}^\pi_{\hat{\mu}} = \hat{T}^\pi_{\hat{\mu}} \hat{V}^\pi_{\hat{\mu}}$. The solution to the fixed point equation

$$V = \hat{T}^\pi_{\hat{\mu}} V \tag{8}$$

can be determined exactly by solving a linear system in n variables by matrix inversion (or approximately by iterating backups): i.e. to perform this version of policy iteration the value function only needs to be maintained at the successor sample points $s'_i \in \mathcal{D}$, even if $\mathcal{S}$ is continuous. This should be contrasted to the typical situation in approximate dynamic programming, in which the Bellman equation cannot be solved exactly in general, and backups followed by projection can diverge. For the same reason the greedy policy can be executed anywhere on $\mathcal{S}$ using knowledge of $\hat{V}^\pi_{\hat{\mu}}(\cdot)$ only at the sample points $s'_i \in \mathcal{D}$ since we can construct the corresponding action-value function at any state-action, via

$$\hat{Q}^\pi_{\hat{\mu}}(s,a) = r(s,a) + \gamma \sum_{i=1}^n \alpha_i(s,a) \hat{V}^\pi_{\hat{\mu}}(s'_i). \tag{9}$$

We refer to a CME such that $\sum_j |\alpha_j(s,a)| \leq 1$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ a *proper CME*. When $\alpha_j(s'_i, a) \geq 0$ and $\sum_k \alpha_k(s'_i, a) = 1$ for all i, j, $\hat{T}^\pi_{\hat{\mu}}$ is the Bellman operator for an MDP defined on the sample $\{s'_i\}_{i=1}^n$, whose dynamics are defined by $P(s'_j | s'_i, a) = \alpha_j(s'_i, a)$. Otherwise $\alpha_j(s'_i, a)$ does not define an MDP, hence we refer to the induced *pseudo-MDP* in general. A theory of pseudo-MDPs, as a method of MDP abstractions, has been developed in Yao et al., 2014. A few important results include the following: The condition that makes CMEs proper allows one to define value functions for policies the usual fashion (with expectation w.r.t. signed measures). The value functions satisfy the analog Bellman equations. If one defines the optimal value function as the fixed-point of the analogue Bellman optimality equation then this optimal value function can be found either by linear programming, or value-/policy-iteration.

Policy Iteration using CMEs These observations motivate a generic model-based policy iteration algorithm for solving MDPs using CMEs: Collect data to learn an approximate proper CME (2) of the transition dynamics; solve the approximate Bellman equation (7) on the samples exactly; Construct the approximation (9) and take the greedy policy $\text{greedy}(\hat{Q}^\pi_{\hat{\mu}})$; iterate if necessary. For clarity this is outlined in pseudocode in Algorithm 1.

Modeling Value Functions Although we assume value functions can be well-approximated in $\mathcal{F}$, we never need to model them in $\mathcal{F}$, i.e. find a weight vector v such that $\langle v, \phi(s) \rangle_{\mathcal{F}} \approx V(s)$. We will see that it is not even necessary to construct $\phi(s)$ for any s, knowledge of the kernel function $L(s,s') = \langle \phi(s), \phi(s') \rangle$ is sufficient. Thus the approximation space $\mathcal{F}$ for V can be a rich function class.

Algorithm 1 Generic model-based policy iteration with CMEs

Input: MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P_1, P, R)$ to interact with; known mean reward function r, known start-state distribution P_1.
Initialize: $Q_0 = r$, $\mathcal{D}_0 = \emptyset$, $\hat{P}^0$, $\pi_1 = \text{greedy}(Q_0)$
Parameters: n_{new}, J
for $k = 1, 2, ...$ **do**
  Data acquisition: Collect n_{new} data points $\mathcal{D}_{\text{new}}$ (e.g. from the policy π_k or an exploratory policy). Aggregate data: $\mathcal{D}_k = \mathcal{D}_{k-1} \cup \mathcal{D}_{\text{new}}$.
  Update dynamics model: Learn the CME $\hat{\mu}(\cdot)$ using the data $\mathcal{D}_k$.
  for $j = 1, 2, ..., J$ **do**
    Policy evaluation: Form estimate $\hat{V}_j$ of V^{π_k} by solving the approximate Bellman Equation $V = \hat{T}^\pi_{\hat{\mu}} V$ (8). Define $\hat{Q}_j(s,a) = r(s,a) + \gamma \sum_{j=1}^n \alpha_j(s,a) \hat{V}_j(s'_j)$.
    Policy improvement: $\pi_k \leftarrow \text{greedy}(\hat{Q}_j)$.
  end for
  $\pi_{k+1} \leftarrow \pi_k$.
end for

Performance Guarantees It is possible to derive bounds on the value of the learned policy in value iteration in terms of the quality of the learned model and how well V^* can be modeled in $\mathcal{F}$. For example we have the following theorem:

Theorem 3.1. *(Grünewälder et al., 2012b, Theorem 3.2, (see also Yao et al., 2014)) After k iterations of value iteration using a proper CME $\hat{\mu}$ we have the following guarantee for the value of the learned policy π_k*

$$\begin{aligned}||V^{\pi_k} - V^*||_\infty \leq \frac{2\gamma}{(1-\gamma^2)} \Big(\gamma^k ||\hat{V}_1 - \hat{V}_0||_\infty + 2||V^* - \tilde{V}^*||_\infty \\ + \sup_{s,a} ||\mu(s,a) - \hat{\mu}(s,a)||_{\mathcal{F}} ||\tilde{V}^*||_{\mathcal{F}} \Big),\end{aligned} \tag{10}$$

where $\hat{V}_k$ is the estimated value function at iteration k, and $\tilde{V}^*$ is any element of $\mathcal{F}$.

Theorem 3.1 applies to any proper estimate of the CME, independently of how it is learned.

3.2 Review of Current Approaches

We now review the current approaches to learning the CME in model-based reinforcement learning algorithms which result in solving finite MDPs.

Kernel-Based Reinforcement Learning The idea of using an induced finite MDP from a model estimate was, to our knowledge, introduced in the Kernel-Based Reinforcement Learning algorithm (KBRL) of Ormoneit and Sen, 2002 who derive an approximate finite Bellman equation induced by a model and solve it.The model can be viewed as approximat-

ing the CME (2) using kernel smoothing (i.e. a Nadaraya-Watson regressor),

$$\hat{\mu}^{KBRL}(s,a) := \sum_{i=1}^{n} \frac{K((s,a),(s_i,a_i))}{\sum_{j=1}^{n} K((s,a),(s_j,a_j))} \phi(s'_i), \quad (11)$$

where K is a smoothing kernel. No loss function is directly specified or optimized. The induced finite transition dynamics are given by $\alpha_i^{KBRL}(s,a) := \frac{K((s,a),(s_i,a_i))}{\sum_{j=1}^{n} K((s,a),(s_j,a_j))}$ and condition (6) holds. The KBRL method is shown to be consistent: the learned value function will converge to the optimal value function with infinite data.

One drawback of the approach is that the size of the state space of the induced MDP equals $|\mathcal{D}| = n$ and planning via a solving the approximate Bellman equation (7) scales poorly with the data. Versions of KBRL with efficient planning have been proposed using stochastic factorization of the transition matrix of the induced finite MDP (da Motta Salles Barreto, Precup, and Pineau 2011) or a cover tree quantization of the state space (Kveton and Theocharous 2012).

Modeling Transition Dynamics with Kernel Least Squares CMEs Grünewälder et al., 2012b suggest a regularized least squares approach to learning the CME (2) applying methods from the literature on learning CMEs (Song, Fukumizu, and Gretton 2013; Grünewälder et al. 2012a). In general, a normed hypothesis class $\mathcal{H} \subset \mathcal{F}^{\mathcal{S}\times\mathcal{A}}$ is chosen and the empirical loss (4) plus a regularization term is minimized,

$$\hat{\mu}^{RLS} := \operatorname*{argmin}_{\mu\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} ||\mu(s_i,a_i) - \phi(s'_i)||^2_{\mathcal{F}} + \lambda||\mu||^2_{\mathcal{H}}. \quad (12)$$

When $\mathcal{H} = \mathcal{H}_K$ is a RKHS (of $\mathcal{F}$-valued functions) with a kernel K on $\mathcal{S}\times\mathcal{A}$ the solution to (12) is $\hat{\mu}^{KRLS}(s,a) = \sum_{j=1}^{n} \alpha_j^{KRLS}(s,a)\phi(s'_j)$ where

$$\alpha_j^{KRLS}(s,a) = \sum_{i=1}^{n} K((s,a),(s_i,a_i))W_{ij} \quad (13)$$

where $\boldsymbol{W} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}$, and $K_{ij} = K((s_i,a_i),(s_j,a_j))$ is the kernel matrix on the data. The approach to guarantee the constraint (6) is simply to threshold and normalize the $\alpha(s'_i,a)$ for all $s'_i \in \mathcal{D}$ and $a \in \mathcal{A}$, however we find in practice it is better to project the weights using fast Euclidean projection to the L1-ball (Duchi et al. 2008), followed by normalization.Unlike KBRL (Ormoneit and Sen 2002) this approach minimizes a loss over a rich hypothesis class.

Factored Linear Action Models The approach of Yao et al., 2014 is to work in the primal and solve (12) for a particular case that $\phi : \mathcal{S} \to \mathcal{F}$ is a feature map to a Hilbert space $\mathcal{F}$ such that $\langle \phi(s), \phi(s') \rangle_{\mathcal{F}} = \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{otherwise} \end{cases}$, i.e. the limit of a Gaussian kernel as the bandwidth tends to zero, but to impose the constraint (6) in the optimization, rather than projection after learning. In our experience this method requires less normalization of the CME to guarantee a contraction map but tends to perform worse in practice, perhaps because the constraint can reduce the hypothesis class so that fewer good models are represented.

4 Compressed CMEs for RL

As discussed in Section 3.2, one drawback of the CME approach is that the size of the induced finite (pseudo-)MDP scales with the amount of data observed so that planning scales poorly. In this section we will present a version of the algorithm of Grünewälder et al., 2012b which is more efficient to learn, and which induces a small compressed MDP on a subset of the data points to ensure efficient planning. We will learn a *compression set* $\mathcal{C} = \{c_1, ..., c_m\} \subseteq \mathcal{S}$ with $m \ll n = |\mathcal{D}|$, and a *compressed CME* $\hat{\mu}^{CMP}(\cdot)$ of the form

$$\hat{\mu}^{CMP}(s,a) = \sum_{j=1}^{m} \alpha_j^{CMP}(s,a)\phi(c_j), \quad (14)$$

so that the induced finite pseudo-MDP is defined on $\mathcal{C}$. Details of learning such a CME are presented in Section 4.2. First we detail the choice of the compression set $\mathcal{C}$.

4.1 Learning the Compression Set

We now show that learning CME transition models suggest a principled means of learning a compression set: our compression set will guarantee the existence of a compressed CME maintaining good guarantees via Theorem 3.1. We first introduce a useful property of compression sets:

Definition Given any set $\mathcal{P} = \{s'_j\}_{j=1}^n$, and any small error δ, a compression set $\mathcal{C} = \{c_j\}_{j=1}^m$ is a *δ-lossy compression* of $\mathcal{P}$, if given any proper CME $\hat{\mu}(\cdot)$ of the form $\hat{\mu}(s,a) = \sum_{j=1}^{n} \alpha_j(s,a)\phi(s'_j)$, there exists a proper CME $\hat{\mu}^{CMP}(s,a) = \sum_{j=1}^{m} \alpha_j^{CMP}(s,a)\phi(c_j)$, which approximates $\hat{\mu}(s,a)$ in $\mathcal{F}$ in the sense that $\sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} ||\hat{\mu}(s,a) - \hat{\mu}^{CMP}(s,a)||_{\mathcal{F}} < \delta$.

Algorithm 2 gives a method to maintain a δ-lossy compression set. The required minimization problem $\min_{\boldsymbol{b}\in\mathbb{R}^m, ||\boldsymbol{b}||_1\leq 1} || \sum_{i=1}^{m} b_i\phi(c_i) - \phi(s'_j)||_{\mathcal{F}}$ can be solved using Lasso (see Appendix A for details). We remark that the feature vectors $\phi(s)$ never need to be explicitly computed.

Algorithm 2 `augmentCompressionSet`$(\mathcal{C}, \delta, \mathcal{P})$

Input: Initial compression set $\mathcal{C} = c_1, ..., c_m$, candidates $\mathcal{P} = s'_1, ..., s'_n$, tolerance δ
for $j = 1, 2, ..., n$ **do**
  if $\min_{\boldsymbol{b}\in\mathbb{R}^m, ||\boldsymbol{b}||_1\leq 1} || \sum_{i=1}^{m} b_i\phi(c_i) - \phi(s'_j)||_{\mathcal{F}} > \delta$ **then**
    Augment compression set: $\mathcal{C} \leftarrow \mathcal{C} \cup s'_j$, $m \leftarrow m+1$
  end if
end for

Theorem 4.1. *Algorithm 2 with $\mathcal{C}$ initialized to $\emptyset$ returns a δ-lossy compression set of $\mathcal{P}$.*

Proof. See Appendix B. The result follows immediately from Lemma B.1, which states that if $\max_{1\leq j\leq n} \min_{\boldsymbol{b}\in\mathbb{R}^m, ||\boldsymbol{b}||_1\leq 1} || \sum_{i=1}^{m} b_i\phi(c_i) - \phi(s'_j)||_{\mathcal{F}} \leq \delta$ then $\{c_1, \ldots, c_m\}$ is a δ-lossy compression set of $\mathcal{P}$. □

A δ-lossy compression $\mathcal{C}$ guarantees the *existence* of a good compressed CME defined over $\mathcal{C}$. We now further show

that δ-lossy compressions imply compressed CMEs which are good for learning MDPs. We begin with a trivial corollary of Theorem 3.1:

Corollary 4.2. *Suppose proper CMEs $\bar{\mu}(\cdot)$ and $\hat{\mu}(\cdot)$ are such that*

$$\sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} ||\hat{\mu}(s,a) - \bar{\mu}(s,a)||_{\mathcal{F}} < \delta. \quad (15)$$

Then after k iterations of value iteration using a $\bar{\mu}(\cdot)$ we have the following guarantee for the value of the learned policy $\bar{\pi}_k$

$$||V^{\bar{\pi}_k} - V^*||_\infty \leq \frac{2\gamma}{(1-\gamma^2)}\Big(\gamma^k||\hat{V}_1 - \hat{V}_0||_\infty + 2||V^* - \tilde{V}^*||_\infty + \sup_{s,a}||\mu(s,a) - \hat{\mu}(s,a)||_{\mathcal{F}}||\tilde{V}^*||_{\mathcal{F}} + \delta||\tilde{V}^*||_{\mathcal{F}}\Big), \quad (16)$$

where $\hat{V}_k$ is the estimated value function at iteration k, and $\tilde{V}^*$ is any element of $\mathcal{F}$.

Thus, suppose that $\hat{\mu}(\cdot)$ is known to be good, then comparing Corollary 4.2 with Theorem 3.1, we only pay an additional term of $\frac{2\gamma\delta||\tilde{V}^*||_{\mathcal{F}}}{(1-\gamma^2)}$ for solving an MDP with the CME $\bar{\mu}(\cdot)$ instead of the CME $\hat{\mu}(\cdot)$. In particular if we take $\bar{\mu}(\cdot)$ to be a compressed CME $\hat{\mu}^{CMP}(\cdot)$ whose existence is guaranteed by a δ-lossy compression then guarantee (16) holds for $\hat{\mu}^{CMP}(\cdot)$. This is an existence proof: in the following section we detail how to learn a compressed CME.

4.2 Learning Compressed CMEs

We now learn a proper compressed CME, denoted $\hat{\mu}^{PCMP}(\cdot) = \sum_{j=1}^m \alpha^{PCMP}(\cdot)\phi(c_j)$, given a compression set $\mathcal{C}$. We first seek a CME $\mu^{CMP}(\cdot)$ of the form (14) in which the $\alpha_j^{CMP}(s,a)$ are of the form (13),

$$\alpha_j^{CMP}(s,a) = \sum_{i=1}^n K((s,a),(s_i,a_i))W_{ij}^{CMP}, \quad (17)$$

i.e. similar to Grünewälder et al., 2012b, but with a compressed representation. $\boldsymbol{W}^{CMP} \in \mathbb{R}^{n\times m}$ is to be optimized to minimize (4). The exact RKHS regularized least squares solution is prohibitively expensive to compute, and so we optimize $\boldsymbol{W}^{CMP}$ using kernel matching pursuit (Vincent and Bengio 2002; Mallat and Zhang 1993) which is an incremental sparse-greedy method of optimizing kernel least squares. In our application matching pursuit maintains a *sparse basis* of kernel functions $\mathcal{B} = \{K(\cdot,(\hat{s}_\ell,\hat{a}_\ell))\}_{\ell=1}^d$ and $d\times m$ weight matrix $\boldsymbol{W}^{MP} = (\boldsymbol{w}_1,...,\boldsymbol{w}_d)^\top$, where for each ℓ, $(\hat{s}_\ell,\hat{a}_\ell)\in\mathcal{D}$, and where $d \ll n$, such that (17) is represented only using the basis $\mathcal{B}$: $\alpha_j^{MP}(s,a) = \sum_{\ell=1}^d K((s,a),(\hat{s}_\ell,\hat{a}_\ell))W_{\ell j}^{MP}$. Matching pursuit incrementally adds bases $K(\cdot,(\hat{s}_\ell,\hat{a}_\ell))$ and weights $\boldsymbol{w}_\ell\in\mathbb{R}^m$ "greedily" – i.e. to maximally reduce the loss (4), $\frac{1}{n}\sum_{i=1}^n||\sum_{\ell=1}^d\sum_{j=1}^m K((s_i,a_i),(\hat{s}_\ell,\hat{a}_\ell))W_{\ell j}^{MP}\phi(c_j) - \phi(s_i')||_{\mathcal{F}}^2$. Given the next basis element, each new weight $\boldsymbol{w}_\ell$ can be found in closed form, and so each addition of a (basis,weight) pair requires a sweep over all candidate basis points. There are some non-standard aspects of the application of matching pursuit to optimize $\boldsymbol{W}^{CMP}$ and details are included in Appendix D, where we demonstrate for instance that $\mathcal{F}$ can be an infinite-dimensional feature space of an RKHS and yet we can still perform matching pursuit efficiently. The greedy incremental approach is suited to the RL setting since we interact with the system to gather new data, which is then explored by matching pursuit to discover new basis functions, but we do not sweep over the full data set at each iteration (see Algorithm 3 for details).

Once matching pursuit has found the basis $\mathcal{B}$ we "backfit" the weights by performing RKHS-regularized least-squares in the primal,

$$\boldsymbol{W}^{CMP} = \operatorname*{argmin}_{\boldsymbol{W}} \frac{1}{n}\sum_{i=1}^n ||err_i(\boldsymbol{W})||_{\mathcal{F}}^2 + \lambda\mathrm{trace}(\boldsymbol{W}\boldsymbol{L}^{CC}\boldsymbol{W}^\top\boldsymbol{K}^\top), \quad (18)$$

where with slight abuse of notation we redefine $\boldsymbol{W}^{CMP}$ to have dimension $d\times m$, and $err_i(\boldsymbol{W}) := \sum_{\ell=1}^d\sum_{j=1}^m K((s_i,a_i),(\hat{s}_\ell,\hat{a}_\ell))W_{\ell j}\phi(c_j) - \phi(s_i')$. The term $\mathrm{trace}(\boldsymbol{W}\boldsymbol{L}^{CC}\boldsymbol{W}^\top\boldsymbol{K}^\top)$ in (18) is the RKHS norm regularizer $||\hat{\mu}^{CMP}(\cdot)||_K^2$. The solution to (18) is

$$\boldsymbol{W}^{CMP} = (\boldsymbol{\Psi}^\top\boldsymbol{\Psi} + \lambda n\boldsymbol{K})^{-1}\boldsymbol{\Psi}^\top\boldsymbol{L}^{DC}(\boldsymbol{L}^{CC})^{-1}, \quad (19)$$

where $K_{jk} = K((\hat{s}_j,\hat{a}_j),(\hat{s}_k,\hat{a}_k))$, $\Psi_{ij} = K((s_i,a_i),(\hat{s}_j,\hat{a}_j))$, $L_{jk}^{DC} = \langle\phi(s_j'),\phi(c_k)\rangle_{\mathcal{F}}$ and $L_{jk}^{CC} = \langle\phi(c_j),\phi(c_k)\rangle_{\mathcal{F}}$, and λ can be cross validated. Computing (19) is efficient since the inverse is of a $d\times d$ matrix. We add a small ridge term to the kernel matrix inverses to prevent numerical instability.

To ensure a contraction we obtain the proper CME $\hat{\mu}^{PCMP}(\cdot)$ by projecting the learned weights to the L1-ball: denoting $\mathrm{proj}(f) := \mathrm{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^m,||\boldsymbol{\beta}||_1\leq 1}||f - \sum_{j=1}^m\beta_j\phi(c_j)||_{\mathcal{F}}$ we set $\boldsymbol{\alpha}^\perp(s,a) := \mathrm{proj}(\hat{\mu}^{CMP}(s,a))$, which can be done using a Lasso (see Appendix A), and finally we normalize,

$$\boldsymbol{\alpha}^{PCMP}(s,a) := \boldsymbol{\alpha}^\perp(s,a)/||\boldsymbol{\alpha}^\perp(s,a)||_1. \quad (20)$$

4.3 Policy Iteration with Compressed CMEs

Our algorithm combines the compression set learning of Section 4.1, CME learning of Section 4.2 and policy iteration using CMEs of Section 3.1 (specifically the CME $\hat{\mu}^{PCMP}(\cdot)$). The algorithm is detailed in Algorithm 3.

5 Experiments

We summarize the experimental results. For reproducibility more detail is given in Appendix C. We compared our compressed CME method to the KBRL algorithm (Ormoneit and Sen 2002) and the full kernel least-squares method of Grünewälder et al., 2012b. We outperformed the full KBRL method, in terms of the quality of the learned policy, so did not compare to the more efficient versions of KBRL which do not report better performance. We report mean results over 10 runs of each experiment, and standard error. Experiments are run on a single-core cluster which is slow: timing results would be 2-3 times faster on a standard desktop. For all MDPs, at each iteration the learners were given 2 trajectories of data: one from the current greedy policy and a second from an ϵ-greedy ($\epsilon = 0.3$) "exploratory" version (defining ρ^{ν_k} in the algorithm as a mixture of those two policies).

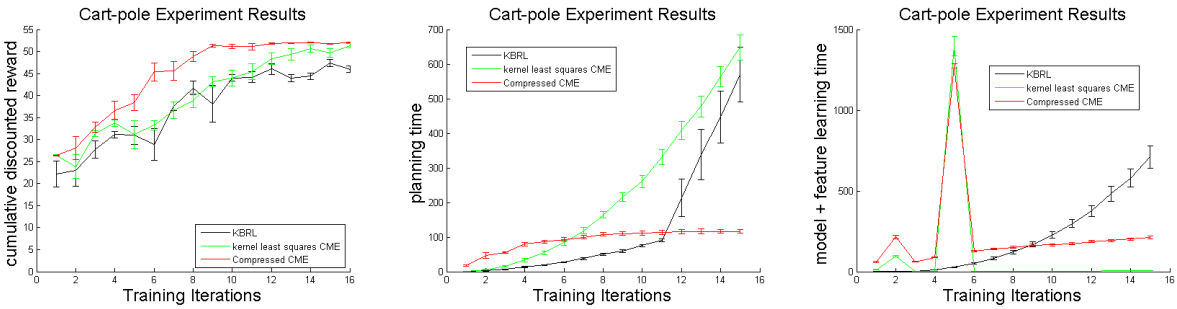

Figure 1: Cart-pole reward and timing results

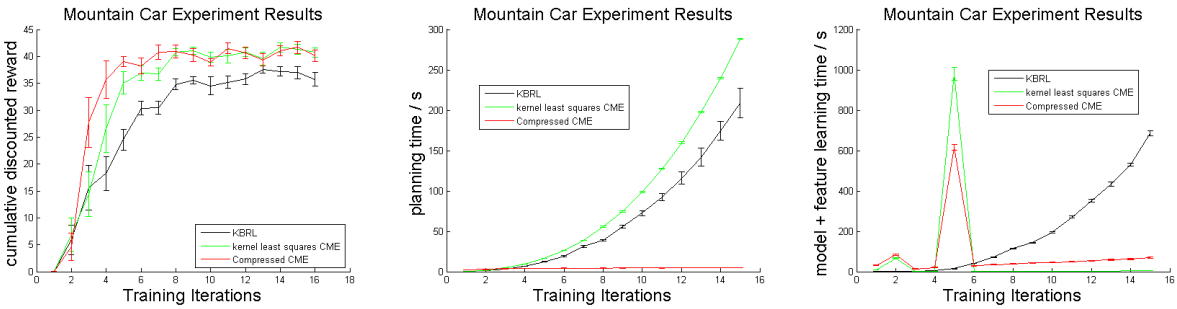

Figure 2: Mountain-car reward and timing results

Benchmarks Our first experiments are the well-known cart-pole and mountain-car benchmark MDPs. These are continuous 2 dimensional state MDPs with 3 actions. The compressed CME approach consistently optimizes the policy in the mountain-car with fewer than 5 iterations (10 data trajectories), and fewer than 10 iterations (20 trajectories) in the cart-pole and performs better than both competitors in terms of policy optimization. We hypothesize that the two least-squares methods out-perform the kernel smoothing KBRL approach since they directly optimize a loss over a rich hypothesis class. Our compressed CME method outperforms the full kernel least squares approach because a more principled projection of the weights (20) can be performed. The compressed CME approach obtained finite MDPs of size less than 100 in the mountain-car and about 400 in the cart-pole (from a total of 3000 data points over 15 iterations), decoupling the amount of data gathered and the size of the induced pseudo-MDP so that planning scales essentially independently of data size, compared to the competitors where planning becomes a bottleneck. In terms of model and feature learning time, the full kernel least-squares approach is surprisingly fast to learn, this is because "full relearns" in which all parameters were cross-validated were performed at iterations 1,2 and 5 (hence the spikes on the graphs) but at other iterations we performed fast online updates to the matrix inverses, using previous optimal parameters. Since model learning is not a bottleneck for KBRL we allowed it to cross-validate all parameters at every iteration; KBRL could be similarly speeded-up by avoiding model selection, with a small degradation in policy quality. Results are shown in Figure 1 and Figure 2.

Simulated Quadrocopter Navigation The third experiment is a simulated Quadrocotper navigation task which uses a simulator (De Nardi 2013) calibrated to model the dynamics of a Pelican$^{\text{TM}}$platform. This is a higher dimensional problem, $\mathcal{S} \subset \mathbb{R}^{13}$ (but effectively reduced to 6-dimensions by the choice of state kernel), $s = (x, y, z, \theta\, \phi, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\theta}, \dot{\phi}, \dot{\psi}, F)$ which consists of platform position $(x, y, z) \in \mathbb{R}^3$, roll, pitch and yaw $(\theta, \phi, \psi) \in \mathbb{R}^3$, associated time derivatives, and thrust F applied to the rotors. The action set is a discretization of 2-dimensional space into 81 actions which represent desired velocity vectors in the (x, y) plane (desired z velocity is fixed at zero). An internal PID controller translates these desired velocities into low level commands issued to the rotors in attempt to attain those velocities. This creates complex dynamics for the system. The platform is initialized at position $(0, 0, -50)$, a target location is defined at coordinates $x_{targ} = (5, 5)$ and we define $r(s, a) = e^{-\frac{1}{50}||x_{targ} - s_{xy}||^2}$.

On this task the compressed CME method is able to learn a near optimal policy, accelerating before hovering around the target point to collect reward. Both least-squares methods out performed the KBRL method, and planning for the compressed CME approach is significantly more efficient. Compression to a finite pseudo-MDP of size 100-150 (from a total of 2400 data points over 12 iterations) was achieved. Results are shown in Figure 3.

6 Summary

We have presented a model-based policy iteration algorithm, based on learning a compressed CME of the transition dynamics, and solving the induced finite pseudo-MDP exactly using dynamic programming. This results in a stable algorithm with good performance guarantees in terms of the transition model error. An analysis of the CME approach suggested a principled means to choose a compression set for

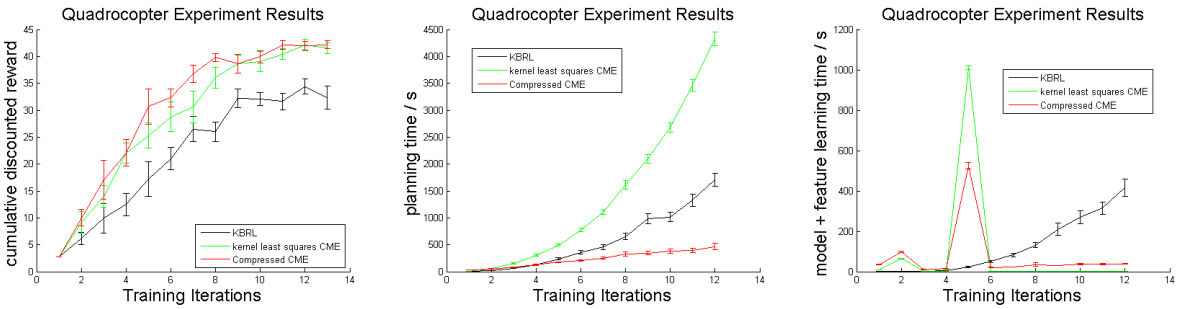

Figure 3: Simulated quadrocopter navigation reward and timing results

the CME, which guarantees an at most small loss in the representation power of the CME, and only a small degradation in performance bound. To learn the CME our approach builds a state-action representation online based on interactions with the system using sparse-greedy feature selection from a data-defined dictionary of kernel functions, enhancing the state-action representation to maximally improve the model. We overcome the planning bottleneck of competitor methods and obtain better policies on a range of MDPs. The compression approach could be used in other related algorithms such as KBRL. Future work could include using other representations of the CME in this context, such as deep neural networks. We remark that it is possible to combine the compression approach with the constrained approach of Yao et al., 2014 which we defer for a longer version.

Appendix

A The Weight Projection via Lasso

Projections of the following form

$$\text{proj}(\boldsymbol{\alpha}) = \operatorname*{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^m, ||\boldsymbol{\beta}||_1\leq 1} ||\sum_{j=1}^m \alpha_j\phi(s_j) - \sum_{j=1}^m \beta_j\phi(s_j)||_{\mathcal{F}}. \tag{21}$$

are used throughout the method to maintain compression sets and project weights to obtain propoer CMEs. Problem (21) can be reduced to:

$$\text{proj}(\boldsymbol{\alpha}) = \operatorname*{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^m, ||\boldsymbol{\beta}||_1\leq 1} ||\boldsymbol{R}\boldsymbol{\beta} - \boldsymbol{R}\boldsymbol{\alpha}||_2.$$

where $\boldsymbol{R}^\top\boldsymbol{R} = \boldsymbol{L}$, and $L_{ij} = \langle\phi(s_i), \phi(s_j)\rangle_{\mathcal{F}}$, which can be solved with Lasso (Tibshirani 1996). Note that the feature vectors do not need to be computed – the Gram matrix is all that is needed. $\boldsymbol{R}$ can be an incomplete (low-rank) Cholesky decomposition (Shawe-Taylor and Cristianini 2004) of $\boldsymbol{L}$ with few rows, so that the effective number of "data points" in the Lasso is small, and therefore efficient. For example we use an incomplete Cholesky factorization with 200 rows in our experiments.

B Proof of Theorem 4.1

We begin with a lemma:

Lemma B.1. *Given* $\{s'_1, s'_2, ..., s'_n\}$, *suppose* $\mathcal{C} = \{c_1, ..., c_m\} \subseteq \mathcal{S}$ *is such that for all* $s'_j \in \mathcal{D}$ *there exists* $\boldsymbol{b} = \boldsymbol{b}(s'_j) \in \mathbb{R}^m$, $||\boldsymbol{b}||_1 \leq 1$ *such that*

$$||\sum_{i=1}^m b_i(s'_j)\phi(c_i) - \phi(s'_j)||_{\mathcal{F}} \leq \delta. \tag{22}$$

Then let $\mu(\cdot)$ *be defined via* $\mu(s,a) = \sum_{j=1}^n \alpha_j(s,a)\phi(s'_j)$, *with* $||\boldsymbol{\alpha}(s,a)||_1 \leq 1$ *for all* $(s,a) \in \mathcal{S}\times\mathcal{A}$. *If we define* $\alpha_i^{CMP}(s,a) = \sum_{j=1}^n b_i(s'_j)\alpha_j(s,a)$ *then* $\sum_{i=1}^m |\alpha_i^{CMP}(s,a)| \leq 1$ *and for all* (s,a)

$$||\mu(s,a) - \hat{\mu}^{CMP}(s,a)||_{\mathcal{F}} < \delta.$$

Proof.

$$\sum_{i=1}^m |\alpha_i^{CMP}(s,a)| = \sum_{i=1}^m |\sum_{j=1}^n b_i(s'_j)\alpha_j(s,a)|$$
$$\leq \sum_{j=1}^n |\alpha_j(s,a)| \sum_{i=1}^m |b_i(s'_j)| \leq 1,$$

and,

$$||\hat{\mu}(s,a) - \hat{\mu}^{CMP}(s,a)||_{\mathcal{F}}$$
$$= ||\sum_{j=1}^n \alpha_j(s,a)\phi(s'_j) - \sum_{i=1}^m \alpha_i^{CMP}(s,a)\phi(c_i)||_{\mathcal{F}}$$
$$= ||\sum_{j=1}^n \alpha_j(s,a)(\phi(s'_j) - \sum_{i=1}^m b_i(s'_j)\phi(c_i))||_{\mathcal{F}}$$
$$\leq \sum_{j=1}^n |\alpha_j(s,a)| \max_j ||\phi(s'_j) - \sum_{i=1}^m b_i(s'_j)\phi(c_i))||_{\mathcal{F}}$$
$$\leq \delta.$$

□

Note that by construction the compression set returned by Algorithm 2 satisfies condition (22) and is therefore a δ-lossy compression. Hence Lemma B.1 directly implies Theorem 4.1.

C Experimental Details

We here give more details of the experiments for reproducibility purposes. We begin with settings common to all MDPs. The horizon of each MDP is 100, so that $n_{\text{new}} = 200$ data points were added at each iteration. To perform planning at each iteration we performed $J = 10$ policy evaluation/improvement steps, before returning to the MDP to collect more data. For the compressed CME the size of the

Algorithm 3 Policy iteration with compressed kernel CMEs

Input: MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, P_1, P, R\}$ to interact with; known reward function R and P_1.
Parameters: Kernel K on $\mathcal{S} \times \mathcal{A}$; feature map $\phi(\cdot)$ on $\mathcal{S}$; compression tolerance δ; policy improvements J; n_{new}; maximum basis dimension d.
Initialize: action-value function e.g. $Q_0 = r$; $\mathcal{D}_0 = \emptyset$; $\pi_1 = \text{greedy}(Q_0)$; $\psi^0(\cdot)$; $n_0 = 0$; $\mathcal{B}_0 = \emptyset$.
for $k = 1, 2, ...$ **do**
  Data acquisition: Collect n_{new} data points $\mathcal{D}_{\text{new}} = \{s_i, a_i, s_i'\}_{i=n_{k-1}+1}^{n_k}$ from behavior policy distribution ρ^{ν_k}; $n_k \leftarrow n_k + n_{\text{new}}$; Aggregate data: $\mathcal{D}_k = \mathcal{D}_{k-1} \cup \mathcal{D}_{\text{new}} = \{s_i, a_i, s_i'\}_{i=1}^{n_k}$.
  Augment compression set with new data: $\mathcal{C}_k = \text{augmentCompressionSet}(\mathcal{C}_{k-1}, \{s_i'\}_{i=n_{k-1}+1}^{n_k}, \delta)$.
  Augment candidate feature dictionary new data: $\mathcal{G}_k = \mathcal{B}_{k-1} \cup \{K(\cdot, (s_i, a_i))\}_{i=n_{k-1}+1}^{n_k}$
  Sparse basis selection: Learn sparse basis $\mathcal{B}_k = \{K(\cdot, (\hat{s}_\ell, \hat{a}_\ell))\}_{\ell=1}^{d_k}$, $d_k \leq d$ from candidates $\mathcal{G}_k$ using matching pursuit; Set $\psi_\ell^k(\cdot) = K((\hat{s}_\ell, \hat{a}_\ell), \cdot)$, $\boldsymbol{\Psi}_k = \left(\psi^k(s_1, a_1), ..., \psi^k(s_{n_k}, a_{n_k})\right)^\top$.
  Backfit CME weights: $\boldsymbol{W}_k^{CMP} = (\boldsymbol{\Psi}_k^\top \boldsymbol{\Psi}_k + \lambda n \boldsymbol{K})^{-1} \boldsymbol{\Psi}_k^\top \boldsymbol{L}^{DC} (\boldsymbol{L}^{CC})^{-1}$ as in (19).
  for $\ell = 1, 2, ..., J$ **do**
    Policy evaluation: Using finite pseudo-MDP dynamics $\boldsymbol{\alpha}^{PCMP}(c_j, a)$ (20) for $a \in \mathcal{A}$, $c_j \in \mathcal{C}$ solve approximate Bellman Equation (7) to obtain estimate $\hat{V}_\ell$ of V^{π_k} at the compression points $c_j \in \mathcal{C}$. Set $\hat{Q}_\ell(s, a) = r(s, a) + \gamma \sum_{j=1}^{|\mathcal{C}_k|} \alpha_j^{PCMP}(s, a) \hat{V}_\ell(c_j)$.
    Policy improvement: $\pi_k \leftarrow \text{greedy}(\hat{Q}_\ell)$.
  end for
  $\pi_{k+1} \leftarrow \pi_k$.
end for

sparse-greedy feature space was constrained to be no greater than $d = 200$. For all 3 methods we performed 5-fold cross-validation over 10 bandwidth parameters to optimize the "input" kernel K on $\mathcal{S} \times \mathcal{A}$. For the two least-squares methods we also cross-validated the regularization parameter over 20 values. The "output" feature map $\phi(\cdot)$ also corresponds to a Gaussian kernel $\phi(s) = L(s, \cdot)$, and the bandwidth of L was chosen using an informal search for each MDP (it is not clear how this parameter can be validated during the algorithm). For planning we set $\gamma = 0.98$, but we report results for $\gamma = 0.99$.

C.1 Cart-Pole Experiment Details

This problem simulates a pole attached at a pivot to a cart, and by applying force to the cart the pole must be swung to the vertical position and balanced. The problem is under-actuated in the sense that insufficient power is available to drive the pole directly to the vertical position, hence the problem captures the notion of trading off immediate reward for long term gain. We used the same simulator as Lagoudakis and Parr, 2003, except here we choose a continuous reward signal. The state space is two dimensional, $s = (\theta, \dot{\theta})$ representing the angle ($\theta = 0$ when the pole is pointing vertically upwards) and angular velocity of the pole. The action set is $\{-50, 0, 50\}$ representing the horizontal force in Newtons applied to the cart. Uniform noise in $[-10, 10]$ is added to each action. The system dynamics are $\theta_{t+1} = \theta_t + \Delta_t \dot{\theta}_t$, $\dot{\theta}_{t+1} = \dot{\theta}_t + \Delta_t \ddot{\theta}_t$ where

$$\ddot{\theta} = \frac{g \sin(\theta) - \alpha m \ell (\dot{\theta})^2 \sin(2\theta)/2 - \alpha \cos(\theta) u}{4\ell/3 - \alpha m \ell \cos^2(\theta)},$$

where $g = 9.8 m/s^2$ is the acceleration due to gravity, $m = 2kg$ is the mass of the pole, $M = 8kg$ is the mass of the cart, $\ell = 0.5m$ is the length of the pole and $\alpha = 1/(m + M)$. We choose $\Delta_t = 0.1s$. Rewards $R(s, a) = \frac{1+\cos(\theta)}{2}$, the discount factor is $\gamma = 0.99$, the horizon is $H = 1000$, and the pole begins in the downwards position, $s_0 = (\pi, 0)$.

The state kernel is a Gaussian $L(s, s') = \exp\left(\frac{1}{2\sigma_{\mathcal{S}}^2}(s - s')^\top \boldsymbol{M}_{\mathcal{S}}(s - s')\right)$ with $\boldsymbol{M}_{\mathcal{S}} = \begin{pmatrix} 1 & 0 \\ 0 & 1/4 \end{pmatrix}$, and the state-action kernel is $K((s, a), (s', a')) = \exp\left(\frac{1}{2\sigma_{\mathcal{S}\times\mathcal{A}}^2}((s, a) - (s', a'))^\top \boldsymbol{M}_{\mathcal{S}\times\mathcal{A}}((s, a) - (s', a'))\right)$ with

$$\boldsymbol{M}_{\mathcal{S}\times\mathcal{A}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/10000 \end{pmatrix}.$$

The output feature map $\phi(s) = L(s, \cdot)$ with $\sigma_{\mathcal{S}} = 0.5$ (chosen by informal search). During model learning we performed 5-fold cross validation over a range of 10 bandwidths in the range $[0.01, 5]$ to optimize $\sigma_{\mathcal{S}\times\mathcal{A}}$. For the compressed CME the tolerance of the compression set selection was set to $\delta = 0.1$, i.e. we use a δ-lossy compression set $\mathcal{C}$ with $\delta = 0.1$.

Cumulative reward of 50 is a near optimal policy in which the pole is quickly swung up and balanced for the entire episode. Cumulative reward of 40 to 45 indicates that the pole is swung up and balanced, but either not optimally quickly, or that the controller is unable to balance the pole for the entire remainder of the episode.

C.2 Mountain-Car Experiment Details

In this problem the agent controls a car located at the bottom of a valley and the objective is to drive to the top of a hill but does not have enough power to achieve this directly and must climb the opposite hill a short distance before accelerating toward the goal (see e.g. Singh and Sutton, 1996). States $s = (x, v)$ are position and velocity, $\mathcal{S} = (-1.2, 0.7) \times (-0.07, 0.07)$, $\mathcal{A} = \{-1, 0, 1\}$ and $r(s, a) = e^{-8(x-0.6)^2}$ and $s_0 = (-0.5, 0)$. Dynamics are $x' = x + v + \epsilon_1$, $v' = v + 0.001a - 0.0025\cos(3x) + \epsilon_2/10$, where ϵ_1, ϵ_2 are Gaussian random variables with standard deviation 0.02. If $x' > 0.6$ then the state is set to $(0.6, 0)$. For a horizon $H = 100$ the optimal return is around 45. Doing nothing receives almost no reward.

The state kernel is a Gaussian $L(s, s') = \exp\left(\frac{1}{2\sigma_{\mathcal{S}}^2}(s - s')^\top \boldsymbol{M}_{\mathcal{S}}(s - s')\right)$ with $\boldsymbol{M}_{\mathcal{S}} = \begin{pmatrix} 1 & 0 \\ 0 & 1/100 \end{pmatrix}$, and the state-

action kernel is $K((s,a),(s',a')) = \exp\left(\frac{1}{2\sigma^2_{\mathcal{S}\times\mathcal{A}}}((s,a)-(s',a'))^\top \boldsymbol{M}_{\mathcal{S}\times\mathcal{A}}((s,a)-(s',a'))\right)$ with

$$\boldsymbol{M}_{\mathcal{S}\times\mathcal{A}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1/100 & 0 \\ 0 & 0 & 1/25 \end{pmatrix}.$$

The output feature map $\phi(s) = L(s,\cdot)$ with $\sigma_{\mathcal{S}} = 0.5$ (chosen by informal search). During model learning we performed 5-fold cross validation over a range of 10 bandwidths in the range $[0.01, 5]$ to optimize $\sigma_{\mathcal{S}\times\mathcal{A}}$. For the compressed CME the tolerance of the compression set selection was set to $\delta = 0.01$, i.e. we use a δ-lossy compression set $\mathcal{C}$ with $\delta = 0.01$.

C.3 Quadrocopter Experiment Details

The state kernel is a Gaussian $L(s,s') = \exp\left(\frac{1}{2\sigma^2_{\mathcal{S}}}(s-s')^\top \boldsymbol{M}_{\mathcal{S}}(s-s')\right)$ with $\boldsymbol{M}_{\mathcal{S}} = diag(\mathbf{1}/25, \mathbf{0}, \mathbf{1}, \mathbf{0}, 0)$ where $\mathbf{1} = (1,1,1)$, $\mathbf{0} = (0,0,0)$ (which essentially reduces the state observations to 6 dimensions), and the state-action kernel is

$$K((s,a),(s',a')) = \exp\left(\frac{1}{2\sigma^2_{\mathcal{S}\times\mathcal{A}}}((s,a)-(s',a'))^\top \boldsymbol{M}_{\mathcal{S}\times\mathcal{A}}((s,a)-(s',a'))\right)$$

with $\boldsymbol{M}_{\mathcal{S}\times\mathcal{A}} = diag(\mathbf{1}/25, \mathbf{0}, \mathbf{1}, \mathbf{0}, 0, \mathbf{1}/100)$. The output feature map $\phi(s) = L(s,\cdot)$ with $\sigma_{\mathcal{S}} = 1$ (chosen by informal search). During model learning we performed 5-fold cross validation over a range of 10 bandwidths in the range $[0.01, 5]$ to optimize $\sigma_{\mathcal{S}\times\mathcal{A}}$. For the compressed CME the tolerance of the compression set selection was set to $\delta = 0.1$, i.e. we use a δ-lossy compression set $\mathcal{C}$ with $\delta = 0.1$.

References

Bach, F. R., and Jordan, M. I. 2005. Predictive low-rank decomposition for kernel methods. In *ICML 2005*, 33–40.

Baird, L. C. 1995. Residual algorithms: Reinforcement learning with function approximation. In *ICML*, 30–37.

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

Bertsekas, D. P. 2011. Approximate policy iteration: a survey and some new methods. *Journal of Control Theory and Applications* 9(3):310–335.

Bertsekas, D. P. 2012. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 4th edition.

Bradtke, S. J., and Barto, A. G. 1996. Linear least-squares algorithms for temporal difference learning. *Machine Learning* 22(1-3):33–57.

da Motta Salles Barreto, A.; Precup, D.; and Pineau, J. 2011. Reinforcement learning using kernel-based stochastic factorization. In *NIPS 2011*, 720–728.

De Nardi, R. 2013. The qrsim quadrotor simulator. Technical Report RN/13/08, Department of Computer Science, University College London, Gower Street, London UK.

Duchi, J. C.; Shalev-Shwartz, S.; Singer, Y.; and Chandra, T. 2008. Efficient projections onto the *l*1-ball for learning in high dimensions. In *ICML 2008*, 272–279.

Grünewälder, S.; Lever, G.; Baldassarre, L.; Patterson, S.; Gretton, A.; and Pontil, M. 2012a. Conditional mean embeddings as regressors. In *ICML 2012*.

Grünewälder, S.; Lever, G.; Baldassarre, L.; Pontil, M.; and Gretton, A. 2012b. Modelling transition dynamics in mdps with rkhs embeddings. In *ICML 2012*.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press.

Kveton, B., and Theocharous, G. 2012. Kernel-based reinforcement learning on representative states. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*.

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

Lever, G., and Stafford, R. 2015. Modelling policies in mdps in reproducing kernel hilbert space. In *AISTATS 2015*.

Mallat, S., and Zhang, Z. 1993. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing* 41(12):3397–3415.

Ormoneit, D., and Sen, S. 2002. Kernel-based reinforcement learning. *Machine Learning* 49(2-3):161–178.

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

Shawe-Taylor, J., and Cristianini, N. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Singh, S. P., and Sutton, R. S. 1996. Reinforcement learning with replacing eligibility traces. *Machine Learning* 22(1-3):123–158.

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

Sutton, R.; Szepesvári, C.; Geramifard, A.; and Bowling, M. H. 2008. Dyna-style planning with linear function approximation and prioritized sweeping. In *UAI 2008*, 528–536.

Tibshirani, R. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)* 58:267–288.

van Hoof, H.; Peters, J.; and Neumann, G. 2015. Learning of non-parametric control policies with high-dimensional state features. In *AISTATS 2015*.

Vincent, P., and Bengio, Y. 2002. Kernel matching pursuit. *Machine Learning* 48(1-3):165–187.

Yao, H.; Szepesvári, C.; Pires, B. A.; and Zhang, X. 2014. Pseudo-mdps and factored linear action models. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement*, 1–9.

D Additional Details of the Feature Selection Process

D.1 Sparse-Greedy Regression with Vector-Valued Matching Pursuit

Our algorithm uses, as a subcomponent, a vector-valued version of the matching pursuit algorithm (Mallat and Zhang 1993). The use of matching pursuit for vector-valued regression is not new (Lever and Stafford 2015) but we provide details for completeness. This adaptation can handle targets in general vector spaces $\mathcal{V}$. Since the algorithm only computes inner products between vectors in the target space $\mathcal{V}$ it can be kernelized, i.e. we can learn an RKHS-valued function. This is a straightforward extension of the scalar case, we derive the method here for clarity.

Suppose we wish to regress a vector-valued function

$$f^* : \mathcal{X} \to \mathcal{V},$$

given a data sample $\mathcal{D} = \{x_i, v_i\}_{i=1}^m$ where $v_i = f^*(x_i) + \epsilon$ where ϵ is zero-mean noise, $f^*(x_i) = \mathbb{E}[V_i|x_i]$. Suppose we are given a *dictionary* $\mathcal{G} = \{g_1, ..., g_n\}$, where $g_i : \mathcal{X} \to \mathbb{R}$, of candidate real-valued functions, and we aim to find an estimate $\hat{f}$ for f^* of the form,

$$\hat{f} = \sum_{i=1}^{D} w^i \hat{g}_i$$

where $\mathcal{B}_D = \{\hat{g}_i\}_{i=1}^D \subseteq \mathcal{G}$ is called the basis and $w^i \in \mathcal{V}$. When $\mathcal{V} = \mathbb{R}$, matching pursuit (Mallat and Zhang 1993) can be used to incrementally build the basis, and we now detail the extension to the vector-valued output case. We build the basis incrementally and for each basis $\mathcal{B}_j$ we form an estimate $\hat{f}^j = \sum_{i=1}^j w^i \hat{g}_i$. We begin with the empty basis $\mathcal{B}_0$ and add new basis elements $\hat{g}_{j+1}$ to greedily optimize the objective. For each estimate we define the residue $\boldsymbol{r}^j$,

$$r_i^j = v_i - \hat{f}_j(x_i) \in \mathcal{V},$$

and pick the $g \in \mathcal{D}$ which minimizes the next residue when added to the current estimate,

$$\begin{aligned} g_{j+1} &= \operatorname*{argmin}_{g \in \mathcal{D}} \min_{w \in \mathcal{V}} \sum_{i=1}^m ||v_i - ((\hat{f}_j + wg)(x_i))||_{\mathcal{V}}^2 \\ &= \operatorname*{argmin}_{g \in \mathcal{D}} \min_{w \in \mathcal{V}} \sum_{i=1}^m ||r_i^j - wg(x_i)||_{\mathcal{V}}^2. \end{aligned}$$

Since $\nabla_w \sum_{i=1}^m ||r_i^j - wg(x_i)||_{\mathcal{V}}^2 = 0$ at the minimum we have,

$$\begin{aligned} 0 &= \sum_{i=1}^m \nabla_w \left(\langle g(x_i)w, g(x_i)w \rangle_{\mathcal{V}} - 2\langle g(x_i)w, r_i^j \rangle_{\mathcal{V}} \right) \\ &= \sum_{i=1}^m 2wg(x_i)^2 - 2g(x_i)r_i^j \\ w^{j+1} &= \left(\sum_{i=1}^m g(x_i) r_i^j \right) / \left(\sum_{i=1}^m g(x_i)^2 \right) \in \mathcal{V} \end{aligned}$$

Then,

$$\begin{aligned} &\sum_{i=1}^m ||r_i^j - w^{j+1} g(x_i)||_{\mathcal{V}}^2 \\ &= \sum_{i=1}^m ||r_i^j||_{\mathcal{V}}^2 - 2\sum_{i=1}^m g(x_i)\langle r_i^j, w^{\min} \rangle_{\mathcal{V}} \\ &\quad + ||w^{\min}||_{\mathcal{V}}^2 \sum_{i=1}^m g(x_i)^2 \\ &= \sum_{i=1}^m ||r_i^j||_{\mathcal{V}}^2 - \frac{2\sum_{i=1}^m g(x_i)\langle r_i^j, \sum_{k=1}^m g(x_k) r_k^j \rangle_{\mathcal{V}}}{\sum_{k=1}^m g(x_k)^2} \\ &\quad + \frac{||\sum_{k=1}^m g(x_k) r_k^j||_{\mathcal{V}}^2}{\sum_{i=1}^m g(x_i)^2} \\ &= \sum_{i=1}^m ||r_i^j||_{\mathcal{V}}^2 - \frac{||\sum_{i=1}^m g(x_i) r_i^j||_{\mathcal{V}}^2}{\sum_{i=1}^m g(x_i)^2} \end{aligned} \tag{23}$$

Thus $\hat{g}_{j+1} = \operatorname{argmax}_{g \in \mathcal{G}} \frac{||\sum_{i=1}^m g(x_i) r_i^j||_{\mathcal{V}}^2}{\sum_{i=1}^m g(x_i)^2}$. Thus at each iteration of matching pursuit we must evaluate $\frac{||\sum_{i=1}^m g(x_i) r_i^j||_{\mathcal{V}}^2}{\sum_{i=1}^m g(x_i)^2}$ for a selection of k dictionary elements (not necessarily all). We have,

$$||\sum_{i=1}^m g(x_i) r_i^j||_{\mathcal{V}}^2 = ||\sum_{i=1}^m g(x_i)(\hat{f}^j(x_i) - v_i)||_{\mathcal{V}}^2$$

For each dictionary element g this can be computed in $O(mj + md + jd)$ where $d = \dim(\mathcal{V})$, and so $O(k(mj + md + jd))$ over k dictionary elements.

It is sometimes useful, at iteration j to "backfit" all the weights $\{w^i\}_{i=1}^j$ by replacing them with the least squares solution: i.e. matching pursuit is used to find the basis but the weights are finally optimized using least squares. Alternatively this can be performed end of the process or several times throughout.

In order to find a compact representation we can also use matching pursuit adaptively by setting a tolerance δ such that the algorithm terminates when it fails to reduce the residue by more than δ. Thus the method will only add features if they significantly reduces the objective.

The output of vector valued matching pursuit is a collection of weights $\{w^i\}_{i=1}^j$ and features $\{\hat{g}_i(\cdot)\}_{i=1}^j$ such that $f^* \approx \sum_{i=1}^j w^i \hat{g}_i$.

Fast Feature Selection For RKHS-Valued Matching Pursuit Performing matching pursuit to regress a function $f : \mathcal{S} \times \mathcal{A} \to \mathcal{F}_L$ where $\mathcal{F}_L$ is an RKHS whose feature map $\phi(s) = L(s, \cdot)$ is high- or even infinite-dimensional can become expensive when the number of data points becomes large. This is because the expansion of each target residue for matching pursuit will have an expansion in the RKHS $\mathcal{F}_L$ in the number of data points, and so computing a single inner product scales with the size of the data, and feature selection by this method would be the bottleneck in Algorithm 3. To solve this problem we map the target data for matching pursuit, i.e. the residues $\mathcal{R}(s_i, a_i, s_i') \in \mathcal{F}_L$

for $(s_i, a_i, s_i') \in \mathcal{D}$, into a lower dimensional subspace of $\mathcal{F}_L$ using an incomplete Cholesky decomposition of the kernel matrix $\boldsymbol{L}$ (Shawe-Taylor and Cristianini 2004). This provides an approximation $\boldsymbol{L} \approx \boldsymbol{R}^\top \boldsymbol{R}$ of the $n \times n$ matrix $\boldsymbol{L}$ where $\boldsymbol{R}$ is a $p \times n$ matrix $\boldsymbol{R} = (\boldsymbol{r}_1, ..., \boldsymbol{r}_n)^\top$, where $p \ll n$ can be chosen or chosen adaptively using a tolerance parameter. This approximation is often excellent (Bach and Jordan 2005) and captures the inner products in $\mathcal{F}_L$ since $\boldsymbol{r}_i^\top \boldsymbol{r}_j \approx L_{ij} = \langle \phi(s_i'), \phi(s_j') \rangle_L$, thus we can "project" the high-dimensional feature vectors $\phi(s_i')$ to $\mathbb{R}^p$ via $\phi(s_i') \mapsto \boldsymbol{r}_i$, approximately preserving inner products. We then have the following approximation for the loss function,

$$\hat{\text{loss}}_\lambda(\boldsymbol{W}) = \frac{1}{n} \sum_{i=1}^n || \sum_{j=1}^n \psi(s_i, a_i)^\top \boldsymbol{w}_j \phi(s_j) - \phi(s_i)||_{\mathcal{F}}^2$$
$$\approx \frac{1}{n} \sum_{i=1}^n || \sum_{j=1}^n \psi(s_i, a_i)^\top \boldsymbol{w}_j \boldsymbol{r}_j^\top - \boldsymbol{r}_i^\top ||^2 \quad (24)$$

Thus if we define the following "projection" of the model residues

$$\boldsymbol{q}_i := \boldsymbol{r}_i^\top - \sum_{j=1}^{n_k} \psi^k(s_i, a_i)^\top \boldsymbol{w}_k^j \boldsymbol{r}_j^\top \in \mathbb{R}^p, \quad (25)$$

and we perform vector-valued matching pursuit using the approximate, low dimensional residue data $\{(s_i, a_i), \boldsymbol{q}_i\}_{i=1}^n$ in p-dimensional Euclidean space we will find find a feature $\psi^{\text{new}}(\cdot)$ and a weight $\boldsymbol{b}^{\text{new}} = \sum_{j=1}^n w_j^{\text{new}} \boldsymbol{r}_j^\top$ such that

$$\sum_{i=1}^n ||\psi^{\text{new}}(s_i, a_i)^\top \boldsymbol{b}^{\text{new}} + \sum_{j=1}^n \psi(s_i, a_i)^\top \boldsymbol{w}_j \boldsymbol{r}_j^\top - \boldsymbol{r}_i^\top ||^2$$

is minimized w.r.t $\psi^{\text{new}}(\cdot) \in \mathcal{G}$ and $\boldsymbol{b}^{\text{new}} \in \mathbb{R}^p$. But then $\psi^{\text{new}}(\cdot)$ and a $b^{\text{new}} := \sum_{j=1}^n w_j^{\text{new}} \phi(s_j')$ are (approximately) optimally greedy for $\hat{\text{loss}}_\lambda(\boldsymbol{W})$ in the sense that

$$\sum_{i=1}^n ||\psi^{\text{new}}(s_i, a_i)^\top b^{\text{new}} + \sum_{j=1}^n \psi(s_i, a_i)^\top \boldsymbol{w}_j \phi(s_j) - \phi(s_i)||_{\mathcal{F}}^2$$

is minimized w.r.t. $\psi^{\text{new}}(\cdot) \in \mathcal{G}$ and $b^{\text{new}} \in \mathcal{F}$. i.e. performing matching pursuit on the low dimensional residues is an equivalent problem up to approximation error of the Cholesky decomposition and the same features will be returned if this decomposition is accurate. In this way d_{new} new features can be added in time $O(d_{\text{new}} p |\mathcal{G}||\mathcal{D}|)$, where p is the rank of the incomplete Cholesky decomposition. In our experiments p was set to $p = 200$ by computational budget and the kernel approximation was found to be almost perfect.