Learning Transition Dynamics in MDPs with Online Regression and Greedy Feature Selection*

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

Ronnie Stafford
University College London
London, UK
ronnie.stafford@gmail.com

John Shawe-Taylor
University College London
London, UK
jst@cs.ucl.ac.uk

Abstract

We present an approach to reinforcement learning in which the system dynamics are modelled using online linear regression between feature spaces, and a compact feature representation for the dynamics model is built incrementally using greedy feature selection. Candidate features are built online using kernels centred at datapoints as they are discovered. We implement the model learning method in a policy iteration scheme. The complexity of each policy iteration (feature learning, model learning, value estimation and policy improvement) is independent of the total amount of data observed, and only linear in the amount of new data added per iteration. The approach therefore scales up to complex problems requiring a huge amount of data to learn well. We validate the approach on benchmark MDPs and simulated quadrocopter navigation.

1 Introduction

1.1 Overview

We consider a *model-based* approach to solving Markov decision processes (MDPs), in which the system dynamics are unknown, and a model is explicitly learned from data collected online during interaction with the system, and used during planning to optimize the policy. In this work we learn a function μ^* which delivers the expected successor feature map, conditioned on a current state-action, $\mu^*(s,a) := \mathbb{E}_{S' \sim P(\cdot|s,a)}[\boldsymbol{\phi}(S')]$ where $\boldsymbol{\phi} : \mathcal{S} \to \mathbb{R}^d$ is a feature map on states (which we assume to be provided and is suitable for modelling the value function). Learning dynamics in this form has been considered in, for example, Grünewälder et al. (2012b) using kernel regression. Major issues include using the methods in an online context, scalability and appropriate representation of state-actions, which we consider here. We attempt to achieve the performance of recent powerful kernel estimators, but with an approach that scales up. We consider using our model as a component in an approximate policy iteration framework (see e.g. Bertsekas, 2012).

We model μ^* as a linear function between two feature spaces by finding an estimator

$$\hat{\mu}(s,a) := \boldsymbol{B}\boldsymbol{\psi}(s,a) \approx \mu^*(s,a), \tag{1}$$

where $\boldsymbol{\psi}(\cdot)$ is a feature map on state-actions. Unlike $\boldsymbol{\phi}(\cdot)$, which is used to model the value function, the feature map $\boldsymbol{\psi}(\cdot)$, used to model the dynamics, is not provided a-priori and we build a compact feature representation, over the course of the policy iteration process. This is achieved by sweeping through a dictionary of candidate features at every model update and greedily selecting those which help to improve the dynamics model. The dictionary of candidate features is not pre-defined, but is data-dependent and built online using kernel functions defined on observed data. The learned representation is compact since features which do not significantly improve the dynamics model are not used, and the complexity of the representation therefore adapts automatically to the complexity

*Appearing at the NIPS 2014 Workshop, Autonomously Learning Robots.

of the problem. By increasing the power of the representation in this data-driven way we maintain a good dynamics model over the whole region we have explored. Since the dynamics model we build is independent of any policy we can efficiently update the model (1) online at each policy iteration $\hat{\mu}(s,a) := \boldsymbol{B}\boldsymbol{\psi}(s,a)$ by stochastic gradient descent, sweeping over a subset of the data collected. We therefore do not have to relearn a new dynamics model after policy improvement. The complexity of each of step of our policy iteration scheme (selecting new features; updating the dynamics model; estimating the value of the current policy; performing policy improvement) is *independent* of the number of total number datapoints discovered, and is only linear in the number of new datapoints added at each iteration.

1.2 Related Work

The approach of Grünewälder et al. (2012b) is similar to ours: they consider learning the dynamics using batch kernel least squares. This approach has strong guarantees but good training data is assumed to be provided in advance from an oracle, which is unrealistic in many RL scenarios. Further, learning the model scales cubicly in the datasize and so is not directly applicable to large problems where more than a few thousand data points are needed to learn complex dynamics. Kroemer and Peters (2011) and Deisenroth and Rasmussen (2011), for example, also present powerful kernel-based models, which scale cubicly in the amount of data processed.

The form of our dynamics model is related to the linear dynamics model considered in Parr et al. (2008), who investigate relationships between linear value models and linear dynamics models, and also suggest feature selection to build the state or state-action representation. However Parr et al. (2008) do not consider learning dynamics, but assume that the true dynamics are known and show that fixed point methods of value estimation are equivalent to a particular implicit representation of dynamics. We are concerned with policy optimization in this type of model, using data collected online. Parr et al. (2008) consider linear representations of the dynamics for a particular policy π in value estimation, whereas in our modular approach we separate the dynamics learning from any particular policy, and learn the dynamics of $(s,a) \to s'$ in order to reuse this knowledge of the dynamics during policy optimization.

1.3 Reinforcement Learning Background

We first recall the basic concepts associated with reinforcement learning (RL) problems. 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 comprises: a *state space* $\mathcal{S}$; an *action space* $\mathcal{A}$; an *initial state distribution* P_1 over $\mathcal{S}$; stationary *transition dynamics distribution* with conditional distribution $P(S_{t+1}|S_t = s_t, A_t = a_t)$ satisfying the Markov property $P(S_{t+1}|S_i = s_i, A_i = a_i, \forall i \leq t) = P(S_{t+1}|s_t = s_t, A_t = a_t)$; a (potentially stochastic) *reward function* $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We denote, for convenience, the mean of the reward function by $r(s,a) = \mathbb{E}[R(s,a)]$, and we use the shorthand $P(S_{t+1}|s_t, a_t) = P(S_{t+1}|S_t = s_t, A_t = a_t)$ and generally denote the successor state of s by s'. Given an MDP an *agent* controls a *trajectory* $\boldsymbol{\xi}_H = (s_1, a_1, s_2, a_2, ..., s_H, a_H)$ through $\mathcal{S} \times \mathcal{A}$ by sequentially selecting the actions $a_t \in \mathcal{A}$ at each time step according to a chosen stationary stochastic *policy* $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ is the set of probability distributions on $\mathcal{A}$. We denote the discounted stationary distribution of state-actions when following a policy π by $\rho^\pi = (1-\gamma)\sum_{t=1}^{\infty} \rho_t^\pi$ where ρ_t^π is the state-action distribution at time t when following π. In *policy optimization*, the agent's goal is to obtain a policy which maximizes the *return* (cumulative discounted reward), $U(\pi) := \lim_{H\to\infty} \mathbb{E}[r_\gamma(\boldsymbol{\xi}_H); \pi]$ where, $r_\gamma(\boldsymbol{\xi}_H) := \sum_{t=1}^{H} \gamma^{t-1} r(S_t, A_t)$ and $\mathbb{E}[\cdot; \pi]$ denotes the expectation with respect to P_1, P and π. Similarly recall the *value function* $V^\pi(s) := \lim_{H\to\infty} \mathbb{E}[r_\gamma(\boldsymbol{\xi}_H)|S_1 = s; \pi]$ and *action-value function* $Q^\pi(s,a) := r(s,a) + \gamma \mathbb{E}_{S'\sim P(\cdot|s,a)}[V^\pi(S')]$. 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 $\pi(s) := \sup_{a\in\mathcal{A}} Q(s,a)$ and denote $\pi = \text{greedy}(Q)$. Obtaining V^π for a given π is known as *value estimation*. It is well known that any V^π satisfies the *Bellman expectation equation*,

$$V^\pi(s) = \mathbb{E}_{A\sim\pi(s)}[r(s,A) + \gamma \mathbb{E}_{S'\sim P(\cdot|s,A)}[V^\pi(S')]] \tag{2}$$

In large MDPs value functions are typically represented in some approximation architecture, $V^\pi(s) \approx \langle \boldsymbol{w}_\pi, \boldsymbol{\phi}(s) \rangle =: \hat{V}^\pi(s)$, where $\boldsymbol{\phi} : \mathcal{S} \to \mathcal{F}_{\boldsymbol{\phi}}$ is a *feature map* on states. In the approx-

Algorithm 1 Policy Iteration with online CME

Input: reward function r; feature map $\boldsymbol{\phi} : \mathcal{S} \to \mathbb{R}^p$; system (MDP) to interact with.
Initialize: Q-function e.g. $Q_0 = r$, $\mathcal{D}_0 = \{\}$, $\hat{\boldsymbol{B}}^0 = \mathbf{0}$, $\pi_1 = \text{greedy}(Q_0)$, $\boldsymbol{\psi}^0(\cdot)$.
Parameters: n, m, learning rates θ, η, ω
for $k = 1, 2, ...$ **do**
  Data acquisition: Collect m data points $\mathcal{D}_{\text{new}}$ from on-policy distribution ρ^{π_k} and exploratory distribution ρ^{ν_k}. Aggregate data: $\mathcal{D}_k = \mathcal{D}_{k-1} \cup \mathcal{D}_{\text{new}}$.
  Feature selection: Obtain new features $\boldsymbol{\psi}^{\text{new}}(\cdot)$, and weights $\boldsymbol{B}_{\text{new}}$ by matching pursuit using dictionary $\mathcal{G}_k$. Update $\boldsymbol{\psi}^k(\cdot) = \begin{pmatrix} \boldsymbol{\psi}^{k-1}(\cdot) \\ \boldsymbol{\psi}^{\text{new}}(\cdot) \end{pmatrix}$ and $\hat{\boldsymbol{B}}_k = (\hat{\boldsymbol{B}}_{k-1}, \theta_k \boldsymbol{B}_{\text{new}})$.
  Update CME dynamics model: Perform n online updates to $\hat{\boldsymbol{B}}_k$ by sweeping over $\mathcal{D}_k$ (6).
  Policy evaluation: Form estimate $\hat{V}_k(s) = \langle \boldsymbol{w}_k, \phi(s) \rangle$ of V^{π_k} by choosing $\boldsymbol{w}_k = \hat{\boldsymbol{w}}_\pi^{\text{BR}}$ (7).
  Policy improvement: Set $Q_k = (1-\omega)Q_{k-1} + \omega \hat{V}_k$ and define greedy policy $\pi_{k+1} = \text{greedy}(Q_k)$.
end for

imate case value estimation entails solving

$$\begin{aligned} \langle \boldsymbol{w}_\pi, \boldsymbol{\phi}(s) \rangle &\approx r(s, \pi(s)) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,\pi(s))}[\langle \boldsymbol{w}_\pi, \boldsymbol{\phi}(S') \rangle] \\ &= r(s, \pi(s)) + \gamma \langle \boldsymbol{w}_\pi, \mathbb{E}_{S' \sim P(\cdot|s,\pi(s))}[\boldsymbol{\phi}(S')] \rangle, \end{aligned} \tag{3}$$

which must be solved approximately since in general no solution in $\boldsymbol{w}_\pi$ to (3) can be found with equality for a given feature map $\boldsymbol{\phi}$ (see e.g. Bertsekas, 2012). In this work we will also utilize a separate feature map $\boldsymbol{\psi} : \mathcal{S} \times \mathcal{A} \to \mathcal{F}_{\boldsymbol{\psi}}$ on state-actions.

2 Policy Iteration Using Online CMEs

2.1 Modelling Dynamics Using Expected Feature Maps

We denote by $\mathcal{D} := \{(s_i, a_i, s_i')\}_{i=1}^m$ our data set, and we suppose that $(s_i, a_i) \sim D$ (we discuss this distribution later) and $s_i' \sim P(\cdot|s_i, a_i)$. Ideally, a data sample is provided from an oracle, but in this work we suppose $\mathcal{D}$ must be collected through interactions with the system and would become available to the learner sequentially. Approximate dynamic programming utilises a function $\mu^* : \mathcal{S} \times \mathcal{A} \to \mathcal{F}_{\boldsymbol{\phi}}$ which delivers the expected successor feature map conditioned on the current state-action, $\mu^*(s, a) = \mathbb{E}_{S' \sim P(\cdot|s,a)}[\boldsymbol{\phi}(S')] \in \mathcal{F}_{\boldsymbol{\phi}}$, sometimes known as the *conditional mean embedding* (CME) of P in $\mathcal{F}_{\boldsymbol{\phi}}$. A canon of work exists providing state-of-the-art methods for learning CMEs, in particular methods using RKHS regularization have been provided Song et al. (2010); Grünewälder et al. (2012a) which optimize the loss,

$$\hat{\ell oss}(\mu) := \sum_{i=1}^m ||\mu(s_i, a_i) - \boldsymbol{\phi}(s_i')||_{\mathcal{F}_\phi}^2. \tag{4}$$

Minimizing (4) is a vector-valued regression problem, and kernel regression approaches have been applied in RL in Grünewälder et al. (2012b), but the complexity scales *cubicly* in the size m of the dataset $\mathcal{D}$, and an $m \times m$ kernel matrix must be stored. In Section 2 we attempt to recover the performance of kernel estimators in RL, but with an approach that scales up.

2.2 A Policy Iteration Algorithm With Online CMEs and Greedy Feature Selection

We now develop our algorithm. We suppose the feature map $\boldsymbol{\phi}(\cdot)$ (for modelling the value function) is given and consider an estimator of the form $\hat{\mu}(s, a) = \hat{\boldsymbol{B}} \boldsymbol{\psi}(s, a)$ where $\boldsymbol{\psi}(\cdot)$ is a feature map on state-actions. Importantly the feature map $\boldsymbol{\psi}(\cdot) = (\psi_1(\cdot), \psi_2(\cdot), ..., \psi_q(\cdot))^\top$ will be constructed incrementally. Our version of policy iteration is outlined in pseudocode given in Algorithm 1. We discus the key steps below.

Data acquisition and exploration is an important problem in RL but not our focus here. We apply the approach of Ross and Bagnell (2012), who suggest to build the data set using a mixture of on-policy

data, from the discounted stationary distribution ρ^{π_k} and data from an exploratory distribution (in experiments we use an exploration policy ν_k at each iteration which is a noisy version of our current policy), and at each iteration k all data is aggregated to form $\mathcal{D}_k = \{(s_i, a_i), s_i'\}_{i=1}^{m_k}$.

Feature selection: This is the key, novel element of our approach. The feature map $\boldsymbol{\psi}(\cdot)$ on state-actions is incrementally constructed at each policy iteration by selecting new feature components $\psi_i(\cdot)$ greedily from a dictionary $\mathcal{G}_k = \{g_1^k(\cdot), g_2^k(\cdot), ...\}$ of candidate functions, where each $g_j^k : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$, in order to improve the dynamics model. At each iteration k we define the *model residue* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathcal{F}_{\boldsymbol{\phi}}$ of $(\hat{\boldsymbol{B}}_{k-1}, \boldsymbol{\psi}^{k-1})$ by $\mathcal{R}(s,a) := \mathbb{E}_{S'\sim P(\cdot|s,a)}[\boldsymbol{\phi}(S')] - \hat{\boldsymbol{B}}_{k-1}\boldsymbol{\psi}^{k-1}(s,a) \in \mathcal{F}_{\boldsymbol{\phi}}$, and its empirical version over a chosen set (a random subsample) $\hat{\mathcal{D}}_k \subseteq \mathcal{D}_k$,

$$\mathcal{R}_{\hat{\mathcal{D}}_k}(s,a,s') := \boldsymbol{\phi}(s') - \hat{\boldsymbol{B}}_{k-1}\boldsymbol{\psi}^{k-1}(s,a) \in \mathcal{F}_{\boldsymbol{\phi}},$$

for each $(s,a,s') \in \hat{\mathcal{D}}_k$. We then use a vector-valued version of the matching pursuit algorithm (Mallat and Zhang, 1993) to select a set of features $\{\hat{g}_1(\cdot), ..., \hat{g}_\ell(\cdot)\} \subseteq \mathcal{G}_k$ and weights $\{\boldsymbol{b}_j \in \mathcal{F}_\phi\}_{j=1}^{\ell}$ such that $\hat{\mathcal{R}}(\cdot) := \sum_{j=1}^{\ell} \boldsymbol{b}_j \hat{g}_j(\cdot)$ approximates the residue $\mathcal{R}(\cdot)$ by minimizing,

$$\mathrm{err}_k(\hat{\mathcal{R}}) := \sum_{(s,a,s')\in\hat{\mathcal{D}}_k} ||\mathcal{R}_{\hat{\mathcal{D}}_k}(s,a,s') - \hat{\mathcal{R}}(s,a)||^2_{\mathcal{F}_\phi}. \tag{5}$$

This is achieved by optimizing sequentially over single features and weights, adding a new feature to maximally reduce (5) each time, until the residue cannot be significantly reduced further. Note then that, for $\theta_k \in [0,1]$,

$$\sum_{(s,a,s')\in\hat{\mathcal{D}}_k} ||\mathcal{R}_{\hat{\mathcal{D}}_k}(s,a,s') - \theta_k\mathcal{R}(s,a)||^2_{\mathcal{F}_\phi} \leq \sum_{(s,a,s')\in\hat{\mathcal{D}}_k} ||\mathcal{R}_{\hat{\mathcal{D}}_k}(s,a,s')||^2_{\mathcal{F}_\phi},$$

since matching pursuit only adds features and weights if they reduce the objective (5). Thus by setting $\boldsymbol{B}_{\mathrm{new}} := (\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_\ell)$ and $\psi_i^{\mathrm{new}}(\cdot) := \hat{g}_i(\cdot)$, and then augmenting the feature representation $\boldsymbol{\psi}^k(\cdot) = \begin{pmatrix} \boldsymbol{\psi}^{k-1}(\cdot) \\ \boldsymbol{\psi}^{\mathrm{new}}(\cdot) \end{pmatrix}$ and $\hat{\boldsymbol{B}}_k = (\hat{\boldsymbol{B}}_{k-1}, \theta_k\boldsymbol{B}_{\mathrm{new}})$, where $\theta_k \in [0,1]$ is a learning rate, we have that $\hat{\boldsymbol{B}}_k\boldsymbol{\psi}(s,a) = \hat{\boldsymbol{B}}_{k-1}\boldsymbol{\psi}^{k-1}(s,a) + \theta_k\boldsymbol{B}_{\mathrm{new}}\boldsymbol{\psi}^{\mathrm{new}}(s,a)$, and therefore,

$$\begin{aligned}\sum_{(s,a,s')\in\hat{\mathcal{D}}_k} ||\boldsymbol{\phi}(s') - \hat{\boldsymbol{B}}_k\boldsymbol{\psi}^k(s,a)||^2_{\mathcal{F}_\phi} &= \sum_{(s,a,s')\in\hat{\mathcal{D}}_k} ||\mathcal{R}_{\hat{\mathcal{D}}_k}(s,a,s') - \theta_k\hat{\mathcal{R}}(s,a)||^2_{\mathcal{F}_\phi} \\ &\leq \sum_{(s,a,s')\in\hat{\mathcal{D}}_k} ||\boldsymbol{\phi}(s') - \hat{\boldsymbol{B}}_{k-1}\boldsymbol{\psi}^{k-1}(s,a)||^2_{\mathcal{F}_\phi}\end{aligned}$$

i.e. the empirical loss (4) of the model is reduced (on $\hat{\mathcal{D}}_k$) by the addition of new weights and features. In this way ℓ new features can be added in time $O(\ell p|\mathcal{G}_k||\hat{\mathcal{D}}_k|)$, where $p = \dim(\mathcal{F}_\phi)$.

We define the dictionary $\mathcal{G}_k$ adaptively at each stage, in a data-driven way. Given a kernel K on $\mathcal{S}\times\mathcal{A}$, when encountering some state-actions $\{(s_i,a_i)\}_{i=1+m_{k-1}}^{m_k}$ during exploration at round k, we set $\mathcal{G}_k = \{g_i(\cdot) = K((s_i,a_i),\cdot)\}_{i=1+m_{k-1}}^{m_k}$, as in kernel matching pursuit (Vincent and Bengio, 2002), in which case we will learn an estimator $(\hat{\mu}(s,a))_i = \sum_{j=1}^{m_k} B_{ij}K((s_j,a_j),(s,a))$, after k rounds, which has the form of a kernel regressor, but here matching pursuit will ensure that the size of this expansion is controlled by only incorporating useful features (most columns of $\boldsymbol{B}$ will be zero). Further we can add functions corresponding to many different kernels to the dictionary, in particular kernels of the same form but different bandwidth, allowing us to learn a representation at various scales throughout the state-action space. The approach is general and the dictionary could include arbitrary real-valued functions. This method is therefore adaptive in two senses: firstly, by selecting new features only if the loss (5) is reduced more than a certain threshold, so that the complexity of the representation adapts to the problem; secondly the feature representation is in terms of kernel functions defined at state-actions that have been discovered. This can be an advantage in RL since a good feature representation can be difficult to choose a-priori.

Model update: The estimators $\hat{\boldsymbol{B}}_k$ have been improved in the feature selection stage with the addition of new weights $\boldsymbol{B}_{\mathrm{new}}$, but all weights can be refined. Since the representation power improves with the addition of new features we can learn complex transition models and are able to sweep over

previous data to maintain a good model over regions previously encountered. We do this by performing stochastic gradient descent on the empirical loos on $\mathcal{D}_k$, $\hat{\ell oss}(\boldsymbol{B}) := \frac{1}{2m_k}\sum_{i=1}^{m_k} ||\boldsymbol{B}\boldsymbol{\psi}(s_i,a_i) - \boldsymbol{\phi}(s_i')||^2 + \frac{\lambda}{2}||\boldsymbol{B}||_{\mathrm{Fr}}^2$, to form a sequence of estimators $\hat{\boldsymbol{B}}_{k-1} = \hat{\boldsymbol{B}}_{k,0}, \hat{\boldsymbol{B}}_{k,1}, ..., \hat{\boldsymbol{B}}_{k,T} = \hat{\boldsymbol{B}}_k$, updated at a randomly chosen datapoint $((s_j,a_j),s_j')$ via

$$\hat{\boldsymbol{B}}_{k,\tau+1} = \hat{\boldsymbol{B}}_{k,\tau} + \eta_\tau^k(\boldsymbol{\phi}(s_j') - \hat{\boldsymbol{B}}_{k,\tau}\boldsymbol{\psi}^k(s_j,a_j))\boldsymbol{\psi}^k(s_j,a_j)^\top \quad - \lambda_\tau^k\hat{\boldsymbol{B}}_{k,\tau}, \tag{6}$$

where $\eta_\tau^k \in [0,1]$ is a learning rate. The complexity of each online sweep over T points is $O(pq_kT)$ where $p = \dim(\mathcal{F}_{\boldsymbol{\phi}})$, $q_k = \dim(\mathcal{F}_{\boldsymbol{\psi}^k})$.

At the **policy evaluation** stage we find a $\boldsymbol{w}_\pi$ such that $V^\pi(s) \approx \langle \boldsymbol{w}_\pi, \boldsymbol{\phi}(s)\rangle =: \hat{V}^\pi(s)$ by minimizing the Bellman residual (Baird, 1995),

$$\boldsymbol{w}_\pi^{\mathrm{BR}} := \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^d} ||\boldsymbol{\Phi}_{\mathcal{C}}\boldsymbol{w} - (\boldsymbol{r} + \gamma\boldsymbol{\Phi}_{\mathcal{C}}'\boldsymbol{w})|| \approx \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^d} ||\boldsymbol{\Phi}_{\mathcal{C}}\boldsymbol{w} - (\boldsymbol{r} + \gamma\boldsymbol{\Psi}_{\mathcal{C}}\hat{\boldsymbol{B}}_k^\top\boldsymbol{w})|| =: \hat{\boldsymbol{w}}_\pi^{\mathrm{BR}}. \tag{7}$$

where $\boldsymbol{\Phi}_{\mathcal{C}} = (\boldsymbol{\phi}(s_1),...,\boldsymbol{\phi}(s_n))^\top$, $\boldsymbol{\Phi}_{\mathcal{C}}' = (\mathbb{E}[\boldsymbol{\phi}(S')|s_1,\pi(s_1)],...,\mathbb{E}[\boldsymbol{\phi}(S')|s_n,\pi(s_n)])^\top$, $\boldsymbol{\Psi}_{\mathcal{C}} = (\boldsymbol{\psi}(s_1,\pi(s_1)),...,\boldsymbol{\psi}(s_n,\pi(s_n)))^\top$, $r_i = r(s_i,a_i)$, and $\mathcal{C} := \{s_1,a_1,s_2,a_2,...,s_n,a_n\}$ is some chosen collection of state-actions. The complexity of performing I iterations of gradient descent utilizing previous value estimates as starting points is $O(|\mathcal{C}|p^2 + Ip^2 + |\mathcal{C}|pq_k)$.

At the **policy improvement** stage, we compute an estimate $\hat{Q}^{\pi_k} \approx Q^{\pi_k}$ via, $\hat{Q}^{\pi_k}(s,a) := r(s,a) + \gamma\langle \boldsymbol{w}_k, \hat{\boldsymbol{B}}_k\boldsymbol{\psi}(s,a)\rangle$ and update Q_k incrementally, $Q_k = Q_{k-1} + \omega(Q^{\pi_k} - Q_{k-1})$, for some chosen $\omega \in [0,1]$, and we then take $\pi_{k+1} = \mathrm{greedy}(Q_k)$. This smoother version of policy iteration was considered by Wagner (2013), and is related in principle to conservative policy iteration (Kakade and Langford, 2002): the idea is that by controlling the updates in this way the new policy will not induce trajectories which leave the region in which the Q function (and also the dynamics model) are accurately modelled, reducing the chance of policy oscillations.

2.3 Learning Feature Dynamics

We also consider using the regression approach to learn the change in the feature map: $\mu^*(s,a) = \mathbb{E}_{S'\sim P(\cdot|s,a)}[\boldsymbol{\phi}(S') - \boldsymbol{\phi}(s)]$ rather than the actual successor state. Thus the data set we use for vector-valued regression of μ^* is $\{(s_i,a_i), \Delta_i := \boldsymbol{\phi}(s_i') - \boldsymbol{\phi}(s_i)\}_{i=1}^m$.

3 Experiments

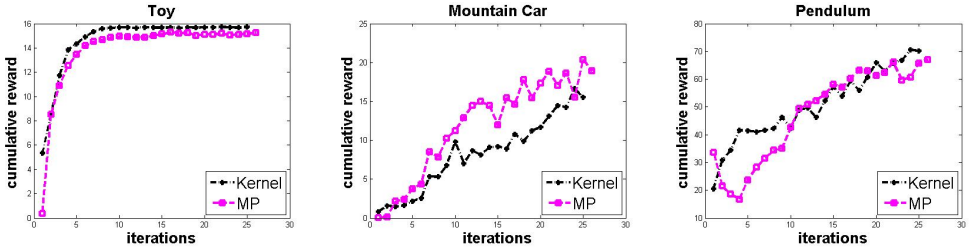

Figure 1: Toy, Mountain Car and Pendulum MDPs

3.1 Benchmarks

We compare our matching pursuit approach, *'MP'*, to an online version of the kernel least squares algorithm of Grünewälder et al. (2012b), *'Kernel'*, in which data is collected online as in our method here, but the dynamics model is relearned every iteration using all accumulated data. To make model learning feasible for *'Kernel'* we collect less data than for *'MP'*. In all experiments we built the dictionary from a collection of Gaussian kernels, centered on all the data we discover, with a range of bandwidths. We learned the change in the feature map as described in Section 2.3.

We first consider a simple **Toy** MDP which is a navigation task on a simple Markov chain on an interval $\mathcal{S} \subset \mathbb{R}$. The second benchmark is the **Mountain Car** problem. The third is the under-actuated **Inverted Pendulum** swing-up problem. Cumulative reward for the three problems is shown in Figure 1. The key advantage of our method - the time taken to learn the model each iteration - is shown in Figure 2: model learning time is almost constant for the matching pursuit version, but cubic for the kernel version, and even on these small problems this becomes an issue.

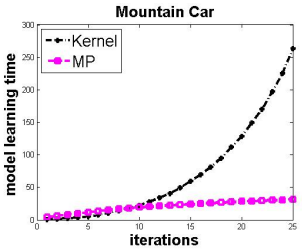

Figure 2: Mountain car model learning time

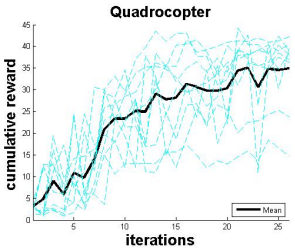

Figure 3: Quadrocopter navigation

3.2 Simulated Quadrocopter navigation

The fourth experiment is a simulated Quadrocotper navigation task which uses a simulator calibrated to model the dynamics of Pelican$^{\text{TM}}$quadrocopter platforms (De Nardi, 2013). $\mathcal{S} \subset \mathbb{R}^{13}$, $s = (x, y, z, \theta\, \phi, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\theta}, \dot{\phi}, \dot{\psi}, F)$ which consists of platform position $s_{xyz} \in \mathbb{R}^3$, platform roll, pitch and yaw $s_{\theta\phi\psi} \in \mathbb{R}^3$, associated time derivatives and the thrust applied to all rotors $F \in \mathbb{R}$. $\mathcal{A} \subset \mathbb{R}^3$ which represent desired velocity vectors for the platform. The simulator mimics the architecture of a real platform such that a PID controller receives these desired velocities at the agent's rate and translates them into low level commands issued directly to the rotor blades at a rate of about $50Hz$, in attempt to attain those velocities, which creates complex dynamics for the system in a state-action space of 16 dimensions. A target location is defined at coordinates x_{targ} such that we define $r(s, a) = e^{-\frac{1}{2\sigma^2}||x - s_{xyz}||^2}$. Average cumulative reward is shown in Figure 3, along with the 10 individual experiment rewards. Our controller learns the complex system dynamics and achieves a policy in which the UAV navigates to the target and hovers around the target point to collect reward.

Acknowledgements

We would like to thank David Silver and Kamil Ciosek for helpful discussions. This work was supported by the European Community Seventh Framework Programme (FP7/2007-2013) under grant agreement 270327 (CompLACS).

References

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *ICML*.

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

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

Deisenroth, M. P. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *ICML*, pages 465–472.

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

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

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *ICML*.

Kroemer, O. and Peters, J. (2011). A non-parametric approach to dynamic programming. In *NIPS*.

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

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*.

Ross, S. and Bagnell, D. (2012). Agnostic system identification for model-based reinforcement learning. In *ICML*.

Song, L., Gretton, A., and Guestrin, C. (2010). Nonparametric tree graphical models. *Journal of Machine Learning Research - Proceedings Track*, 9:765–772.

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

Wagner, P. (2013). Optimistic policy iteration and natural actor-critic: A unifying view and a non-optimality result. In *NIPS*, pages 1592–1600.