Adaptable replanning with compressed linear action models for learning from demonstrations

Clement Gehring **Leslie Pack Kaelbling** **Tomas Lozano-Perez**
Department of Electrical Engineering and Computer Sciences
Massachusetts Institute of Technology
{gehring,lpk,tlp}@csail.mit.edu

Abstract: We propose an adaptable and efficient model-based reinforcement learning approach well suited for continuous domains with sparse samples, a setting often encountered when learning from demonstrations. The flexibility of our method originates from the approximate transition models, estimated from data, and the online replanning approach proposed. Together, these components allow for immediate adaptation to a new task, given in the form of a reward function. The efficiency of our method comes from two approximations. First, rather than representing a complete distribution over the results of taking an action, which is difficult in continuous state spaces, it learns a linear model of the *expected* transition for each action. Second, it uses a novel strategy for compressing these linear action models, which significantly reduces space and time for learning models, and supports efficient online generation of open-loop plans. The effectiveness of these methods is demonstrated in a simulated driving domain with a 20-dimensional continuous input space.

Keywords: Reinforcement learning, learning from demonstration, model predictive control, model-based

1 Introduction

In many robotics problems, there is opportunity to gather data off-line that characterizes the transition dynamics of a domain, but the reward function is not determined until the robot is deployed in the world. For example, in self-driving cars, it is possible to gather dynamics data off-line, on a test-track, but detailed choices about target speeds, closeness to nearby vehicles, destination, and even rules of the road will vary considerably during the deployment of the car.

We address such problems in this paper by proposing a general replanning approach for MDPs, in which a transition model is estimated off-line from sampled transitions from an unknown policy, and then using them online to generate local plans based on the current state and objective.

In this context, we assume that the robot plans only with the provided data with no opportunities for learning later. During evaluation, the robot is expected to quickly compute next actions while adapting to new objectives with minimal delay. We assume the current objective is made available to the robot and takes the form of a reward function which can be evaluated at any time.

In contrast with the typical reinforcement learning setting, our method does not seek an optimal global reactive policy $\pi\colon \mathcal{S} \rightarrow \mathcal{A}$ directly, but, instead, seek a series of approximately optimal local open-loop plans. This strategy reduces the expressiveness required of the policy or value representation while also making each individual planning problem easier. To solve an MDP, we define an agent's behaviour by following a model predictive control (MPC) approach, where a new local plan is generated at every time step to compensate for the local nature of our planning approach. We summarize the high-level structure of our replanning agent in Algorithm 1.

Our contributions are two-fold. First, we present an efficient method for fitting linear action models, a composable model for estimating expected outcomes. Our method uses a novel compressed, factored form of such models which allow for a previously intractable number of features. This opens up the possibility of using powerful but large feature sets approximating reproducing kernel

2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland.

Input: sample transitions $\mathcal{D}$
// off-line processing of data to generate a model
$m \leftarrow \text{FIT}(\mathcal{D})$
// evaluation under potentially changing objectives
while *not finished* **do**
  $s \leftarrow$ GET-CURRENT-STATE()
  $R \leftarrow$ GET-REWARD-FN() // where $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
  $\pi \leftarrow \text{PLAN}(s, R, m)$
  EXECUTE($\pi(0)$)
end

Algorithm 1: A high-level summary of our replanning from learned models approach.

Hilbert space methods, something which had been impractical up to now. Second, we formulate a novel stochastic trajectory optimization problem. We offer an efficient algorithm that exploits both the compositionality and the factored form of our models which results in a fast and practical optimization algorithm for open-loop plans.

We validate our approach by first showing that using our models to estimate state values results in performance similar to that of a model-free temporal difference learning algorithm [1], despite the approximations used. Then, we show our planning approach performs better in a high dimensional continuous racetrack domain when compared to both an analogous model-based, closed-loop value iteration approach using the same model and a model-free approach, batch Q-Learning [2], while seamlessly handling changing reward functions and generating over 6 plans per second.

The idea of using local planning for reinforcement learning has been successfully applied to various reinforcement learning domains [3, 4], but required a simulator to sample results of taking actions in desired states. To our knowledge, this approach has never been applied to learned models.

Our choice of learning expected transitions is motivated by previous work on linear action models (LAM) in which their expressivity, efficiency of learning and efficiency of inference have been leveraged. They have successfully been used in the context of approximate policy iteration (API) [5], approximate value iteration (AVI) [6] and composing modelled meta-actions [7]. We extend previous work by providing a novel low-dimensional approach for estimating linear action models, leveraging the low-rank structure of the state space.

This structure has been previously found and successfully learned in different areas of reinforcement learning, such as in the least-square temporal difference (LSTD) learning algorithm [8] and a second-order temporal difference learning algorithm [9], resulting in significant savings in memory and computation. Finally, low-rank structure was exploited in the parameter matrix of the successor representations, leading to an efficient temporal difference like algorithm [10]. Previous work on this idea enabled sub-quadratic algorithms, opening up previously impractical cases.

2 Background

Framework and Notation We consider the problem of solving an unknown discounted Markov decision process (MDP) $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, q_0, \gamma\}$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}(s, a, s')$ defines the probability of transition from state s to s' given action a, $\mathcal{R}(s, a)$ is the reward received for executing action a in state s, q_0 is a probability distribution over start states and $\gamma \in [0, 1)$ is the discount factor. We limit ourselves to MDPs where planning is done over discrete actions, all of which can be executed in any given state. We use the notation S_t, A_t, R_t to describe the random variables defining the state, the action and the reward, respectively, at time t. Additionally, we use a colon notation as subscript to signify all indices in an interval, inclusively, e.g., $A_{1:t}$ represents $A_1, A_2, \ldots, A_t$. We also use the short hand notation $\mathbb{E}\left[\cdot \mid a_{1:t}, s_0\right]$ in place of $\mathbb{E}\left[\cdot \mid A_{1:t} = a_{1:t}, S_0 = s_0\right]$.

Given an open-loop plan π and a start state s_0, we define a time dependent Q-value function, Q, and its corresponding value function, V, in the following way:

$$Q(t, a; \pi, s_0) = \mathbb{E}\left[\sum_{\tau=t}^{H-1} \gamma^{\tau-t} R_\tau \;\middle|\; A_0 = a, s_0, \pi\right]$$
$$V(t; \pi, s_0) = \mathbb{E}\left[Q(t, a; \pi, s_0) \mid s_0, \pi\right],$$

that is, the expected discounted returns from time t, where action a is executed, to some time horizon H where all other time steps follow plan π and the robot starts in some state s_0 at time step 0. From here, we define an optimal plan from some state s as being a plan π^* which maximizes $V(0; \pi, s)$.

Action Models In order to model the environment, we consider linear functions of non-linear features of states, $\phi \colon \mathcal{S} \to \mathbb{R}^n$, where distributions over next states are encoded as expectations over the value these features. Formally, we have

$$\mathbf{F}_{a_t} \phi(s_t) = \mathbb{E}\left[\phi(S_{t+1}) \mid a_t, s_t\right],$$

where $\mathbf{F}_{a_t} \in \mathbb{R}^{n \times n}$ models the transitions of the MDP under action a_t. Similarly, we model the reward function as a linear function of the same non-linear features, giving us $\mathbf{w}_a^\top \phi(s) = \mathcal{R}(s, a)$,where $\mathbf{w}_a$ is the set of linear coefficients encoding $\mathcal{R}(\cdot, a)$. We refer to a pair $(\mathbf{w}_a, \mathbf{F}_a)$ as a linear action model (LAM).

At first glance, taking a linear approach might appear overly restrictive but with appropriate features, rich families of functions can be encoded. It was shown that, following the reproducing kernel Hilbert space (RKHS) formulation, linear action models can effectively be applied to a wide range of inference and reinforcement learning problems [11, 12, 13]. The idea that expectations over linear features can encode probability distribution is at the core of these approaches. In this work, we use randomly sampled Fourier basis [14] to approximate an RKHS approach with Gaussian kernels.

In addition to being computationally convenient, linear action models allow predictions about future states to be composed together while still accounting for stochastic transitions. Doing so allows us to efficiently evaluate the expected states given a sequence of actions. Formally, given a set of LAM, $\{\mathbf{F}_a : a \in \mathcal{A}\}$, and primitive actions a and b, we can can compute the LAM for executing a followed by b as the matrix product $F_{ab} = F_b F_a$. More generally, we can compose transitions on the expected features as follows:

$$\mathbb{E}\left[\phi(S_{t+1}) \mid a_{1:t}, s_0\right] = \mathbf{F}_{a_t} \mathbb{E}\left[\phi(S_t) \mid a_{1:t-1}, s_0\right] = \mathbf{F}_{a_{1:t}} \phi(s_0),$$

where we use $\mathbf{F}_{a_{1:t}}$ to describe the matrix product $\mathbf{F}_{a_t} \mathbf{F}_{a_{t-1}} \dots \mathbf{F}_{a_1}$. We exploit this property to evaluate the value of a given plan π by observing

$$V(0; \pi, s_0) = \sum_{t=0}^{H-1} \gamma^t \mathbb{E}\left[R_t \mid \pi, s_0\right] = \mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t \mathbf{w}_{a_t}^\top \mathbf{F}_{A_0:A_t} \phi(s_0) \;\middle|\; \pi\right], \tag{1}$$

where the expectation in Eq. 1 is over the actions, allowing random action choices. Note that this equation can be re-written recursively, a property which will be exploited later.

3 Fitting Linear Action Models

In this work, we minimize our model's one-step squared prediction error on the training data. To leverage similarities between actions, we opt for a locally weighted approach rather than keeping transitions of different actions separate. To do so, given a kernel function k defining the similarity between two actions, we define our fitted LAM $(\hat{\mathbf{w}}_a, \hat{\mathbf{F}}_a), a \in \mathcal{A}$ as:

$$\hat{\mathbf{w}}_a = \arg\min_{\mathbf{w}} \sum_{(s_t, a_t, r_t, s_{t+1}) \in \mathcal{D}} k(a, a_t) \left(r_t - \mathbf{w}^\top \phi(s_t)\right)^2 \tag{2}$$

$$\hat{\mathbf{F}}_a = \arg\min_{\mathbf{F}} \sum_{(s_t, a_t, r_t, s_{t+1}) \in \mathcal{D}} k(a, a_t) \parallel \phi(s_{t+1}) - \mathbf{F}\phi(s_t) \parallel_2^2, \tag{3}$$

where $\mathcal{D}$ is a set of sampled transition and we define $\phi(s) = \mathbf{0}$ when s is a terminal state.

The question of what defines a good cost function for model fitting in model-based reinforcement learning is still an open question. However, the one-step prediction error, in the linear case, has a proven relation to the error in value function estimates where small prediction error will result in small Bellman error in the value function [15]. This, combined with our empirical results, lead us to believe that the one-step prediction error is sufficient for our setting.

4 Compressed Linear Action Models (CLAM)

We propose a novel method to fit and store linear action models which we leverage to efficiently find good plans. We consider a compressed, approximate version of LAMs, offering savings in computational time and memory for both the fitting and evaluation.

We derive our approach by relating the data to a locally weighted least-squares approximation of the corresponding LAMs. Given transition tuples (s_t, a_t, r_t, s'_t), we define Φ and Ψ to be the matrices with the predecessor and the successor states organized row-wise, respectively, and the vectors $\mathbf{A}$ and $\mathbf{r}$ to be the actions and the sampled rewards, i.e.,

$$\Phi = \begin{bmatrix} \phi(s_0)^\top \\ \phi(s_1)^\top \\ \vdots \end{bmatrix}, \Psi = \begin{bmatrix} \phi(s'_0)^\top \\ \phi(s'_1)^\top \\ \vdots \end{bmatrix}, \mathbf{A} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \end{bmatrix}, \mathbf{r} = \begin{bmatrix} r_0 \\ r_1 \\ \vdots \end{bmatrix}.$$

We avoid the $O(n^2)$ complexity inherent in LAMs through a low-rank approximation. This is done by finding an embedding of the predecessor states. We do this efficiently by incrementally building a truncated singular value decomposition (SVD) of Φ, following the approach introduced by Brand (2006). The result is an orthogonal embedding for the states in which it is computationally convenient to evaluate expected transitions and their corresponding rewards.

As defined currently, the compressed states, $\Phi \approx \mathbf{U}\Sigma\mathbf{V}^\top$, would allow us to efficiently perform regression over all states, but additional work is required to generate action specific models. We do so by generating an action specific embedding, $\mathbf{U}_a\Sigma_a\mathbf{V}_a^\top$, incorporating the kernel value between each sampled action and some action a by rediagonalizing the row-wise scaled matrix, $diag(K(\mathbf{A}, a))\Phi$. The procedure used can be found in the supplementary material. This approach is significantly more efficient than recomputing the whole embedding from scratch and allows models for new actions to be generated at when needed.

The new embedding allows us to quickly solve the locally weighted regression problem defined in Eq. 2 and 3. Given an action a, this is done with the following equations:

$$\hat{\mathbf{F}}_a = \Psi^\top diag(K(\mathbf{A}, a))\mathbf{U}_a\Sigma_a^{-1}\mathbf{V}_a^\top$$
$$\hat{\mathbf{w}}_a = \mathbf{r}^\top diag(K(\mathbf{A}, a))\mathbf{U}_a\Sigma_a^{-1}\mathbf{V}_a^\top.$$

Additionally, in this work, we modify the inverse Σ^{-1} in order to achieve an L_2-norm regularization of the LAMs. This is done by replacing the inverse singular value σ_i^{-1} by $\sigma_i/(\sigma_i^2 + \lambda)$, where $\lambda \in [0, \infty)$ is the regularization parameter used to adjust the weight on the L_2 penalty.

For computational and memory efficiency, the $n \times n$ matrix $\hat{\mathbf{F}}_a$ is never explicitly computed, and is instead kept in a partially factored form, where $\hat{\mathbf{U}}_a = \Psi^\top diag(K(\mathbf{A}, a))\mathbf{U}_a$ and $\hat{\mathbf{r}}_a = \mathbf{r}^\top diag(K(\mathbf{A}, a))\mathbf{U}_a$ are pre-computed. The final CLAMs consist of two $n \times k$ matrices and a vector of inverse singular values of size k.

5 Planning with CLAMs

5.1 Stochastic Plan Optimization (SPOPT)

We propose an optimization formulation for finding good open-loop plans. That is, given a fixed horizon H, for a state s, we seek a plan π^* such that

$$\pi^* = \arg\max_\pi V(0; \pi, s).$$

Performing this optimization over sequences of actions presents a difficult search problem. To simplify the problem, we formulate a differentiable V by considering a probabilistic description of plans, allowing first-order derivatives to guide the search. This means we define a stochastic open-loop plan as a set of probability distributions over actions, one for each time step up to a horizon H. That is, for a plan π and some time $0 \leq t < H$, we define $\pi(t, \cdot)$ to be a proper probability distribution over actions. This makes the expected outcomes a differentiable function of π, which can be easily locally optimized.

Finally, in order to make the optimization problem unconstrained, we avoid a tabular parametrization of π, and, instead, opt for a soft max encoding of the probabilities. Formally, for parameters $\eta \in$

$\mathbb{R}^{H\times|\mathcal{A}|}$, we have $\pi(t,a) = exp(\eta_{t,a})/\sum_{b\in\mathcal{A}} exp(\eta_{t,b})$. This insures that, for all possible values of η, the corresponding π is a proper stochastic plan.

We solve the planning problem with a normalized gradient ascent of $V(0;\pi,s)$, leveraging the fact that the gradient with respect to $\pi(t,\cdot)$ can easily be computed if $Q(t,\cdot;\pi,s)$ is known. Therefore, the main computational cost of computing gradients lies in estimating Q. As such, we focus our efforts in doing so efficiently, proposing a novel dynamic programming approach.

Estimating Q with LAMs To understand the derivation of our estimation algorithm, it will be helpful to first consider the non-compressed LAM version of our approach. Given LAMs $\{\mathbf{F}_a : a\in\mathcal{A}\}$ and $\{\mathbf{w}_a : a\in\mathcal{A}\}$, features $\phi\colon \mathcal{S}\to\mathbb{R}^n$, and start state $s_0\in\mathcal{S}$, we outline the recursive nature of the time dependent Q-values as defined earlier by defining two new sets of important variables, $\hat{\phi}_t\in\mathbb{R}^n$ and $\alpha_t\in\mathbb{R}^n$, for $0\le t<H$. Here, $\hat{\phi}_t$ represents the expected features at time t following some plan π and α_t represents the remaining expected discounted value attributed to each feature at time t. Both are recursively defined with

$$\hat{\phi}_0 = \phi(s_0);\quad \hat{\phi}_t = \sum_a \pi(t-1,a)\hat{\mathbf{F}}_a\hat{\phi}_{t-1},$$

$$\alpha_H^\top = \mathbf{0};\quad \alpha_t^\top = \sum_a \mathbf{w}_a^\top + \gamma\pi(t,a)\alpha_{t+1}^\top\hat{\mathbf{F}}_a,$$

where the ϕ's represent the expected features moving forward, and, the α's represent the future values of each features propagating backwards. Given this structure, one could follow a dynamic programming approach to compute these quantities efficiently. The result can be used to evaluate the Q-values given some time step t with the following equation:

$$\hat{Q}(t,a;\pi,s_0) = \left(\mathbf{w}_a^\top + \gamma\alpha_{t+1}^\top\hat{\mathbf{F}}_a\right)\hat{\phi}_t.$$

Planning in a Low-Rank Embedding Using the fact that CLAMs are factored and low-dimensional, we can derive a new, mathematically equivalent algorithm for computing the time dependent Q-values. We do this by defining new intermediate quantities, $\alpha_{a,t},\beta_{a,t}\in\mathbb{R}^k$, where k is the rank of the approximations. Similar to before, the vectors $\alpha_{a,t}$ represent the value of features in the low-rank embedding for action a while $\beta_{a,t}$ can be seen as representing the value of the expected features in the embedding for action a at time t. For clarity, we assume each CLAM has the same rank, though this is not necessary. More formally, given the action specific SVDs $\{(\mathbf{U}_a,\Sigma_a,\mathbf{V}_a) : a\in\mathcal{A}\}$, we relate $\alpha_{a,t},\beta_{a,t}$ to $\alpha_t,\hat{\phi}_t$:

$$\beta_{a,t} = \mathbf{V}_a^\top\hat{\phi}_t;\quad \alpha_{a,t}\mathbf{V}_a^\top = \mathbf{w}_a^\top + \gamma\alpha_{t+1}^\top\mathbf{F}_a$$

This allows us to reparameterize the LAMs such that all necessary operations can be performed in the size of the low-rank embedding. Our new models are represented by $\{\mathbf{K}_{a,b} : (a,b)\in\mathcal{A}\times\mathcal{A}\}$ and $\{\boldsymbol{\theta}_a : a\in\mathcal{A}\}$ and relate to the previous form as

$$\boldsymbol{\theta}_a = r_a^\top diag(K(\mathbf{A},a))\mathbf{U}_a$$

$$\mathbf{K}_{a,b} = \mathbf{V}_a^\top\Psi^\top diag(K(\mathbf{A},a))^\top\mathbf{U}_b.$$

Refactoring the recursive equations used to compute $\hat{Q}(t,a;\pi,s_0)$, we get

$$\alpha_{a,t}^\top = \left(\sum_b \pi(t+1,b)\alpha_{b,t+1}^\top\mathbf{K}_{b,a} + \boldsymbol{\theta}_a^\top\right)\Sigma_a^{-1}$$

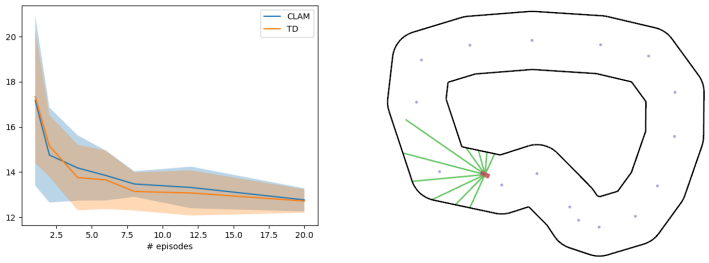
$$\alpha_{a,H-1}^\top = \boldsymbol{\theta}_a^\top\Sigma_a^{-1}$$

$$\beta_{a,t} = \sum_b \pi(t-1,b)\mathbf{K}_{a,b}\Sigma_b^{-1}\beta_{b,t-1}$$

$$\beta_{a,0} = \mathbf{V}_a^\top\phi(s_0).$$

Finally, to evaluate our estimate, we use $\hat{Q}(t,a;\pi,s_0) = \alpha_{t,a}^\top\beta_{t,a}$.

It is important to note that the resulting equations always remain in the dimension of the embeddings. The final runtimes for evaluating a plan are $O(Hk^2|\mathcal{A}|^2+nk)$ for the embeddings approach, $O(Hnk|\mathcal{A}|)$ for using the CLAMs without planning in the embedding, and $O(Hn^2|\mathcal{A}|)$ for the full LAM approach. Since, in practice, $k\ll n$, the compressed models and the corresponding planning algorithm see considerable speed ups. It is important to note that planning in the embedding results in an algorithm which is quadratic in the number of possible actions so this approach is not suited to problems with very large action sets.



(a) (b)

Figure 1: (a) The root means squared error of the estimated values. The standard deviation is reported as the shaded area. (b) The car domain and the observations used.

Determinization of Plans Given the soft max parametrization of π, the gradient ascent of V risks wasting a significant amount of time collapsing the probability distributions to ones and zeros. We note that the value of a stochastic plan π can be improved, or, in the case of a tie, made no worse, by greedily forcing $\pi(t, a^*) = 1$ at some time step t, where $a^* = \arg\max_{a \in \mathcal{A}} Q(t, a; \pi, s_0)$. If done sequentially with respect to t, the intermediate quantities $\beta_{t,a}$ and $\alpha_{t,a}$ can be updated efficiently, keeping the $\hat{Q}$ estimates valid and allowing this step to be repeated. To accelerate planning, we employ a rather aggressive termination condition and compensate by determinizing the plans. The exact procedure used is provided in the supplementary material.

6 Experiments

Mountain Car We first validate our compressed models by examining the estimated values derived by CLAMs in a policy evaluation setting. That is, in the case where only one LAM is generated, corresponding to the expected transition under a fixed policy. For this purpose, we consider the well studied Mountain Car domain [17]. The state consists of a two-dimensional vector, the position and speed of the car. The agent can applying a force either in the positive or negative direction, or staying idle, and must escape a valley. For training, we consider trajectories obtained by following a ϵ-random energy-pumping policy, executing a random action with probability $\epsilon = 0.2$ and otherwise applying a force in the direction the car is moving.

We compare our approach to a batch implementation of the temporal difference learning algorithm (TD) which was run until convergence, reusing provided samples repeatedly. For both methods, we used 4000 random Fourier features and meta-parameters were optimized following a coarse grid-search and used a discount $\gamma = 0.99$. Additional details can be found in the supplementary material.

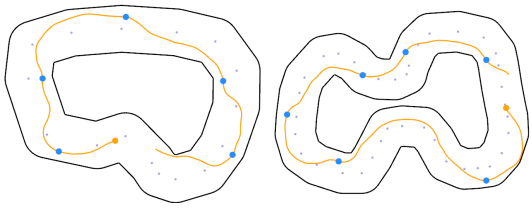
Both methods performed comparably on the value estimation problem. Note that the error is not expected to reach zero given the feature approximation. Figure 1a visualizes the error as more data is provided to each method. The results were averaged over 30 runs. Both methods shared the same seeds in order to ensure the same features and data were used.

Continuous Race Track We further explore the validity of our approach on a more complex car environment with LIDAR-like observations. In this domain, an agent is tasked with driving on a track. Driving off the track causes the episode to terminate, removing the opportunity for future rewards. Training trajectories are generated by having a human control the car. The car dynamics follow that of [18][Sec. 13.1.2.1] with the exception that our agent controls acceleration rather than speed directly. See the supplementary material for more details on the dynamics. Additionally, the environment is made stochastic by the addition of Gaussian noise $\mathcal{N}(\mathbf{0}, diag([0.1, 0.15\pi/4]))$ to the agent's actions. The agent is given 9 discrete actions corresponding to cross-product of the max, min and idle throttle and the max, min, and idle steering inputs.

As an added difficulty, the agent does not receive the state of the car, but, instead, receives observations from a simulated 2D LIDAR. The LIDAR has 10 beams uniformly distributed in front of the car, covering the front $180°$. A beam reports the distance to the closest point on the track which intersects the beam, up to some max distance 1.0. Additionally, each beam reports the relative speed

(a) Track 1 (b) Track 2 (c) Track 3 (d) Track 4 (e) Track 5

Figure 2: Visualization of the tracks and training trajectories.



(a) Track 1 (b) Track 2

Figure 3: Examples of SPOPT's trajectory with repeated lane switches. Blue dot represents the reward switching side, starting from the left.

along the beam at the point it touches or at the max distance, which ever is closest. The result is a 20 dimensional domain with partial observability. Figure 1b provides a visualization of the domain. A total of 12000 random Fourier features were used in these experiments. Additional details can be found in the supplementary material.

To evaluate our approach under changing objectives, we define two reward functions, both requiring the car to move forward in the track. The *left*/*right* reward function requires that the car be on the left/right side of the track, receiving a reward of $\max(-1, \min(1, v))$ when doing so, otherwise receiving $\max(-1, \min(0, v))$. As a consequence, the reward is bounded in $[-1, 1]$ and is only negative if the car reverses and can only be > 0 if moving forward on the correct side. Finally, exiting the track results in a reward of zero and termination of the episode.

We first investigate the value of SPOPT as a planning procedure by comparing its performance to an approximate value iteration (AVI) algorithm, an analogous model-based approach using the same CLAM. We also compare with a batched version of linear Q-Learning, updating on all samples at once until convergence. Both comparison algorithms are run until convergence using the same random features as SPOPT. For Q-Learning, several values for the learning rate are tried and the best solution is reported. It is important to note that all three methods use the same features and choose actions by maximizing over linear Q-values. As a consequence, any differences between these approaches are strictly due to the planning procedure (i.e., gradient optimized local plans, dynamic programming optimized global plan, model-free Q-Learning.)

Track #	# samples	SPOPT	AVI	Q-Learning
1	1785	$\mathbf{1040.4 \pm 1.8}$	678.1 ± 115.8	0.9 ± 1.3
2	1038	$\mathbf{1020.4 \pm 20.2}$	845.3 ± 101.7	22.3 ± 36.6
3	1789	$\mathbf{934.6 \pm 128.9}$	720.9 ± 124.7	0.0 ± 0.1
4	646	512.2 ± 143.5	$\mathbf{907.2 \pm 9.0}$	700.0 ± 178.7
5	2637	85.6 ± 32.0	153.7 ± 69.9	3.6 ± 2.9
1	all samples	$\mathbf{722.3 \pm 131.5}$	245.4 ± 86.8	-0.1 ± 0.0
2	all samples	$\mathbf{1001.2 \pm 14.6}$	277.2 ± 80.5	-0.1 ± 0.0
3	all samples	$\mathbf{687.1 \pm 100.4}$	219.2 ± 76.1	-0.1 ± 0.0
4	all samples	$\mathbf{1043.2 \pm 4.3}$	233.0 ± 69.6	-0.1 ± 0.0
5	all samples	$\mathbf{216.3 \pm 108.9}$	56.9 ± 29.9	-0.1 ± 0.0

Table 1: Comparison of each planning method, across tracks, of the total reward accrued over 60 seconds or until a crash. The reported values correspond to the total reward accrued over 60 seconds or until a crash. The interval reported corresponds to the 0.95 confidence interval.

We compare the quality of the final policies executing them for up to 60 seconds. Figure 2 visualizes the tracks and demonstrations used. Additionally, given the similarities of the domain across tracks, we consider a the case where all five datasets are concatenated. Table 1 provides the total reward obtained while using the *left* reward function.

Overall, SPOPT demonstrates a notable edge over both AVI and Q-Learning. SPOPT appears to be capable of handling the concatenation of the datasets while AVI and Q-Learning fail. We observe a slight decrease in SPOPT's performance in three of the five tracks and an increase in the other two. We conjecture that this difference in behaviour between the planning methods, when comparing the aggregate dataset case and the track specific one, to be due to the local-planning nature of SPOPT. The intuition is that only nearby (expected) states are considered by SPOPT, making it is less vulnerable to 'bad' states from other tracks that would otherwise lead to sub-optimal decisions propagating through the whole global policy.

While AVI had a modest performance with the task specific trajectories, Q-Learning was not able to converge to a good solution in either case and mostly found policies which either crashed immediately or remained mostly stationary. Q-Learning, as well as temporal difference learning in general, are known to have convergence issues [19] when trained on off-policy data which could explain the poor performance. Additionally, we ran the same experiment with a neural net implementation, i.e., DQN [20], and observed the same behaviour.

We note that tracks 4 and 5 have unique difficulties. Track 4 has significantly less data than the other tracks, while track 5 has a self-intersection. Since the concatenated data greatly boosts SPOPT's performance on track 4, we believe that the low sample count is responsible for the initial poor results. Additionally, across all three methods, track 5 has seen poor performance which is likely caused by its unique 8-shape and ambiguous observations it produces.

Finally, we investigate qualitatively SPOPT's behavior under changing reward functions. Since AVI and Q-Learning both plan by incorporating the reward function, both require a full replanning step, making them inflexible and ill-suited for this setting. Along with the source code[1], we provide a short video of an arbitrary track being generated, the demonstrations being collected and the resulting SPOPT policy under changing rewards. Figure 3a and 3b provide visualizations of two additional examples of SPOPT's behaviour under successive changes in objective.

We've found SPOPT to be robust to the tracks and trajectory generated, successfully driving several laps without exiting the track even under changing reward function. SPOPT was able to successfully drive and lane switch despite having sub-optimal demonstrations and few examples. The occasional observed failures, in the majority of cases, can be explained by the agent entering a state which is vastly different from the provided data, e.g., the agent driving perpendicular to the track wall, a situation human drivers typically avoid.

7 Conclusion

In this work, we have presented a novel replanning approach for tackling MDPs with unknown dynamics aimed at a learning from demonstration setting. We show that our SPOPT approach outperforms an analogous value iteration and model-free approach in the quality of the solutions learned while offering superior flexibility with regard to changing objectives, allowing for near immediate adaptation to changes in reward function.

Acknowledgments We gratefully acknowledge support from NSF grants 1420316, 1523767, and 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; from Honda Research; and from Draper Laboratory. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

[1] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

[1] https://github.mit.edu/gehring/CLAM-SPOPT

[2] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[3] C. R. Mansley, A. Weinstein, and M. L. Littman. Sample-based planning for continuous action markov decision processes. In *ICAPS*, 2011.

[4] A. Weinstein and M. L. Littman. Open-loop planning in large-scale stochastic domains. In *AAAI*, 2013.

[5] H. Yao and C. Szepesvári. Approximate policy iteration with linear action models. In *AAAI*, 2012.

[6] H. Yao, C. Szepesvári, B. A. Pires, and X. Zhang. Pseudo-mdps and factored linear action models. In *Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2014 IEEE Symposium on*, pages 1–9. IEEE, 2014.

[7] J. Sorg and S. Singh. Linear options. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 31–38. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[8] C. Gehring, Y. Pan, and M. White. Incremental truncated lstd. In *International Joint Conference on Artificial Intelligence*, 2016.

[9] Y. Pan, A. M. White, and M. White. Accelerated gradient temporal difference learning. In *AAAI*, pages 2464–2470, 2017.

[10] C. A. Gehring. Approximate linear successor representation. In *Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2015.

[11] S. Grunewalder, G. Lever, L. Baldassarre, M. Pontil, and A. Gretton. Modelling transition dynamics in mdps with rkhs embeddings. *ICML*, 2012.

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

[13] G. Lever, J. Shawe-Taylor, R. Stafford, and C. Szepesvári. Compressed conditional mean embeddings for model-based reinforcement learning. In *AAAI*, pages 1779–1787, 2016.

[14] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

[15] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 752–759. ACM, 2008.

[16] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, 2006.

[17] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[18] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[19] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

8 Appendix

Function `GetActionSVD`(*action kernel* K*, sampled actions* $\mathbf{A}$*, svd of predecessor states* $\Phi = \mathbf{U}\Sigma\mathbf{V}^\top$*, an action* a)
- $w \leftarrow K(\mathbf{A}, a)$
- $Q, R \leftarrow qr(diag(w)\mathbf{U})$
- $U', \Sigma', {V'}^\top \leftarrow svd(R\Sigma)$
- $U_a \leftarrow U'Q$
- $\Sigma_a \leftarrow \Sigma'$
- $V_a^\top \leftarrow {V'}^\top \mathbf{V}^\top$
- **return** $U_a, \Sigma_a, V_a^\top$

Algorithm 2: SVD algorithms for finding action specific embeddings.

while π *not converged* **do**
- **for** $t = 0$ *to* $H - 1$ **do**
  - $a^* = \arg\max_a \hat{Q}(t, a; \pi, s_0)$
  - force $\pi(t, a) = 1$ if $a = a^*$, else $\pi(t, a) = 0$
  - update $\hat{\beta}_{t+1,b} \forall b \in \mathcal{A}$
- **end**
- **for** $t = H - 1$ *to* 0 **do**
  - $a^* = \arg\max_a \hat{Q}(t, a; \pi, s_0)$
  - force $\pi(t, a) = 1$ if $a = a^*$, else $\pi(t, a) = 0$
  - update $\hat{\alpha}_{t-1,b} \forall b \in \mathcal{A}$
- **end**

end

Algorithm 3: An efficient determinzation algorithm.

8.1 Parameters for mountaincar

Our implementation of CLAM used 4000 random Fourier bases sampled to approximate a Gaussian kernel with a width of $0.2 \times$ (state-range) and used a Gaussian kernel on actions with a width of 0.5. CLAMs were estimated with a maximum truncated rank $k = 800$ but further truncated the rank to $k = 100$ when evaluating states. Additionally, the models were regularized with $\lambda = 0.2$.

8.2 The race track domain

The car's state is a four dimensional vector defined by its 2D position, it's orientation and speed. The car has parameters $L = 0.1$,, defining its length, and $\rho = 1.0$, defining the damping of speed, modelling friction. For a car in state $q = [x \quad y \quad \theta \quad v]^\top$ and given input $u \in \mathbb{R}^2$, the dynamics of the system follow

$$\dot{q} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \frac{v\tan(u[1])}{L} \\ u[0] - \rho v \end{bmatrix},$$

where $u[0]$ corresponds to the throttle and $u[1]$ corresponds to the angle of the steering wheels with limits $u[0] \in [-1, 1]$ and $u[1] \in [\pi/8, \pi/8]$. The simulation was run at 20 Hz using odeint [**?**]. Control was also run at 20 Hz, requiring a new action every 0.05 seconds.

In this set of experiments, SPOPT used 12000 random Fourier bases approximating a Gaussian kernel with width 0.5 for both LIDAR distance and relative speed measurements. The CLAMs used $\lambda = 0.6$ with a max rank $k = 400$ and a Gaussian action kernel with width 0.5 times the action range. The optimization was performed over a horizon $H = 20$ with the models further truncated to rank $k = 40$. Plans were optimized with a gradient step size of 0.1 with termination when the plan's value improves less than 10^{-4} or if 20 gradient steps were done, which ever happens first. The resulting plan is further improved by three determinization sweeps. Parameters were chosen following a coarse grid search, balancing good, robust solution and fast planning.