Metric Learning for Reinforcement Learning Agents

Matthew E. Taylor, Brian Kulis, and Fei Sha
Lafayette College, taylorm@lafayette.edu
University of California, Berkeley, kulis@eecs.berkeley.edu
University of Southern California, feisha@usc.edu

ABSTRACT

A key component of any reinforcement learning algorithm is the underlying representation used by the agent. While reinforcement learning (RL) agents have typically relied on hand-coded state representations, there has been a growing interest in *learning* this representation. While inputs to an agent are typically fixed (i.e., state variables represent sensors on a robot), it is desirable to automatically determine the optimal relative scaling of such inputs, as well as to diminish the impact of irrelevant features. This work introduces HOLLER, a novel distance metric learning algorithm, and combines it with an existing instance-based RL algorithm to achieve precisely these goals. The algorithms' success is highlighted via empirical measurements on a set of six tasks within the mountain car domain.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Miscellaneous

General Terms

Algorithms, Performance

Keywords

Reinforcement Learning, Distance Metric Learning, Autonomous Feature Selection, Learning State Representations

1. INTRODUCTION

In *Reinforcement Learning* (RL) problems, an agent must learn to select sequences of actions to maximize a reward signal. The agent's decision process is state-dependent — the effects of an action will depend on the agent's location in an environment. The agent's state representation is a critical component in a successful agent, but state representations are typically designed by a human domain expert. The goal of this paper is to introduce a robust method to allow more autonomy in designing state representation, allowing the agent to scale dimensions of the state representation, as well as to potentially ignore irrelevant dimensions.

There has been some exciting recent work on learning to construct or scale state variables (c.f., proto-value functions [10]) but such methods typically assume a model of the task is known. Other work focuses on the placement and tuning of individual basis functions (c.f., learning where to place kernels [2]). In contrast, this work assumes that 1) the agent must efficiently sample the state space and construct its representation on-line and 2) the agent should learn a metric that should generalize across the entire state space, not just the region explored.

Rather than constructing new state variables, we assume that the state variables provided to the agent are sufficient to learn the current task, but that we do not know their relative weighting. For example, consider a robot that has a laser range finder that reads distances in meters and a sonar that reads distances in feet. It is likely that the two state variables will need to be scaled differently to accurately integrate their information. Likewise, if an agent is provided both its speed in meters/second and its acceleration in meters/second^2, the relative importance of these two variables on its estimate of location will need to be treated very differently.

Traditionally, state variables are scaled by normalizing all state variables to have the same range (e.g., $[-1, 1]$). For instance, consider the CMAC [1] function approximator, a type of tile coding used successfully in the mountain car domain [14]. CMACs can take an arbitrary groups of continuous state variables and lay infinite, axis-parallel tilings over them; a continuous state space is discretized while maintaining the capability to generalize via multiple overlapping tilings. However, the number of tiles and width of the tilings are hardcoded by a domain expert, which necessitates knowing both the ranges (to normalize) and relative importance of the different state variables (to determine the spacing and number of tiles per dimension).

This work shows that it is possible to use *distance metric learning*, a popular supervised learning technique, to scale and select state variables automatically from data gathered via agent experience. Experiments show that our theoretically grounded on-line metric learning can result in significantly improved learning in a set of RL tasks situated in the mountain car domain. Our hope is that this work will encourage additional research into the integration of metric learning and RL, as well as to provide a powerful tool to help automatically determine effective state representations.

2. BACKGROUND

This section first introduces Reinforcement Learning, the setting for the paper. Next, Fitted R-MAX is discussed, an instance-based RL algorithm that will be used in this paper's experiments. Last, an introduction to distance metric learning provides background to understand HOLLER, our novel learning algorithm.

2.1 Reinforcement Learning

Reinforcement learning problems are typically framed as *Markov decision processes* (MDPs) defined by the 4-tuple $\{S, A, T, R\}$.

Cite as: Title, Author(s), *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

An agent perceives the current *state* of the world $s \in S$ (possibly with noise). Tasks are often episodic: the agent executes actions in the environment until it reaches a terminal or goal state, at which point the agent is returned to a starting state. The set A describes the *actions* available to the agent, although not every action may be possible in every state. The *transition function*, $T : S \times A \mapsto S$, takes a state and an action as input and returns the state of the environment after the action is performed. The agent's goal is to maximize its reward, a scalar value defined by the *reward function*.

A learner chooses which action to take in a state via a policy, $\pi : S \mapsto A$. π is modified by the learner over time to improve performance, defined as the expected (discounted) total reward. Instead of learning π directly, many RL algorithms instead approximate the action-value function, $Q : S \times A \mapsto \mathbb{R}$, which maps state-action pairs to the expected real-valued return [16]. In tasks with small, discrete state spaces, Q and π can be fully represented in a table. As the state space grows, using a table becomes impractical, or impossible if the state space is continuous. Agents in such tasks typically factor the state using *state variables* (or *features*), so that $s = \langle x_1, x_2, \ldots, x_n \rangle$. In such cases, RL methods use *function approximators*, such as artificial neural networks or tile coding, where parameterized functions representing π or Q are tuned via supervised learning methods. The parameterization and bias of the function approximator define the state space abstraction, allowing observed data to update a region of state-action values rather than a single state/action value.

2.2 Fitted R-Max

The experiments in this paper focus on integrating a learned distance metric with Fitted R-MAX, an instance-based RL algorithm [7]. Fitted R-MAX approximates the action-value function, Q, for large or infinite state spaces by constructing an MDP over a small (finite) sample of states $X \subset S$. For each sample state $\mathsf{x} \in X$ and action $a \in A$, Fitted R-MAX estimates the dynamics of the transition function, $T(\mathsf{x}, a)$, using all available data for action a. The data from multiple nearby states will need to be integrated and generalized as it is unlikely that points in a continuous state space will be sampled enough to approximate all action transitions. A probability over predicted successor states in S, $T(\mathsf{x}, a)$, is first approximated. The distribution of successor states is then approximated with a distribution of states in X, resulting in a MDP defined over a finite size (X) that is formed based on data from the environment (S). Q is then approximated via dynamic programming.

For the purposes of the current work, the most important feature of Fitted R-MAX is that when T and R are estimated for a point x, data from nearby points are averaged together, weighted by their relative distances. That is, recorded instances that are (spatially) closer to x are assumed to be more predictive than instances further away. Rather than assuming that the similarity between points in the state space is Euclidean, this work learns a distance metric for Fitted R-MAX to use. A full description of Fitted R-MAX and its implementation can be found elsewhere [7].

2.3 Distance Metric Learning

Distance metric learning is a core machine learning problem that attempts to learn an appropriate distance function for a given task. Because distances or similarities are used in a variety of tasks — including clustering, similarity searches, and many classification algorithms — there has been significant interest in the design of algorithms for tuning distance functions. Typically these algorithms are at least partially supervised; in addition to the data, the algorithm receives constraints for the desired distance metric. Examples include constraints of the form "points x and y should have a small/large distance" or "points v and w should have a smaller distance than points v and x."

Metric learning algorithms typically attempt to construct a transformation of the data (either linear or non-linear) such that the constraints are satisfied after applying a standard distance function such as the Euclidean distance to the transformed data. The most popular approach is to learn a linear transformation of the data; these methods are often called *Mahalanobis metric learning* methods, and is the approach we employ in this work (c.f., [4, 5, 6, 19, 22]). These methods are desirable in that they show good generalization performance on a variety of problems, including in vision, text, and music domains (c.f., [3, 15]).

Recently, there has been interest in applying metric learning over large-scale data, or in cases when the standard methods that process a large set of constraints in a batch mode are inadequate. Such *online* algorithms instead process a single constraint at a time, and are designed to give comparable performance as compared to their offline counterparts. There has been recent theoretical progress in proving regret bounds for online learning methods, which provide worst-case guarantees on the performance of an online algorithm as compared to any corresponding offline algorithm [13, 23]. We pursue an online approach in this paper to avoid the computational cost of repeatedly applying offline learning methods to our data.

3. LEARNING THE DISTANCE METRIC

Algorithm 1 summarizes the process of learning and using a distance metric in an RL agent. There are three main steps which will be detailed in the following sections:

1. Collect data while the agent explores the environment.
2. Decide which states are "more similar," based on the relatedness of agent transitions.
3. Use state relatedness to calculate a distance metric: states which have similar transitions should be closer than states which have dissimilar transitions.

3.1 Collecting Data

Algorithm 1 is the top-level algorithm. It first initializes an agent (lines 1–4) and then has it interact with its environment for a single episode (lines 5–11), collecting data to be used for distance metric learning. Lines 12–31 consider triples of vectors, where a vector is defined by a pair of states which the agent has moved between (i.e., the difference between s' and s). Lines 18 and 19 consider sets vectors recorded at similar times (e.g., +/- *NumPts* actions).We restrict the vectors to be temporally similar under the assumption that transitions which occur in rapid succession are likely to be more similar than transitions that happen at very different times. This assumption is domain dependent, but will often be true, particularly when *NumPts* is set so that these vectors are also close spatially. However, even in "well behaved" domains there will be regions of the state space where this assumption will be violated (e.g., an agent may often move without obstruction, but be constrained when adjacent to a wall).

We only consider sets of three vectors $\langle v, w, x \rangle$ which have the same action (line 22), as transitions for different actions may be dissimilar. The similarities between vectors v and w, and between vectors v and x are calculated on lines 23 and 24, as discussed in the following section. Lines 27 and 30 add the triple to the set of current constraints, which are in the form "v is more similar to x than v is to w." Finally, after all the data from an episode has been processed, the distance metric is updated with the set of constants.

On lines 33 and 34, the algorithm can decide if more data needs to be collected. For instance, if any W_a has changed significantly

Algorithm 1 Main Algorithm (η)

```
1:  $\pi \leftarrow$ random policy
2:  # initialize the dist. metric for each action
3:  $\forall a \in A$, $W_a \leftarrow$ Identity matrix (i.e., Euclidean distance)
4:  $i \leftarrow 0$
5:  $s \leftarrow$ initial state        # Begin an episode
6:  repeat
7:      Execute $a = \pi(s)$
8:      Observe $r$ and $s'$
9:      Save tuple $V_i \leftarrow (s, a, s')$
10:     $s \leftarrow s'$
11:     $i \leftarrow i + 1$
12: until $s$ is a terminal state        # the episode ends
13: for $j \in \{0, \ldots, i-1\}$ do
14:     # get vector for transition between state $s_j$ and $s'_j$
15:     $v \leftarrow V_j.s' - V_j.s$        #the vector from s to s'
16:     $a \leftarrow V_j.a$        # the action in question
17:     $C_a \leftarrow \emptyset$        # Set of constraints used to update $W_a$
18:     for $k \in \{j - \mathit{NumPts}, \ldots, j + \mathit{NumPts}\}$ do
19:         for $l \in \{j - \mathit{NumPts}, \ldots, j + \mathit{NumPts}\}$ do
20:             $w \leftarrow V_k.s' - V_k.s$        # transition k vector
21:             $x \leftarrow V_l.s' - V_l.s$        # transition l vector
22:             if $(a = V_k.a = V_l.a)$ and $(v, w, x$ are distinct$)$ then
23:                 $re_w \leftarrow$ CALCRELATEDNESS$(W_a, v, w)$
24:                 $re_x \leftarrow$ CALCRELATEDNESS$(W_a, v, x)$
25:                 if $re_w > re_x$ then
26:                     # Relatedness(v,w) $>$ Relatedness(v,x)
27:                     $C_a \leftarrow C_a \cup \langle v, w, x \rangle$
28:                 else
29:                     # Relatedness(v,x) $>$ Relatedness(v,w)
30:                     $C_a \leftarrow C_a \cup \langle v, x, w \rangle$
31:     # update the distance metric
32:     $W_a \leftarrow$ HOLLER$(W_a, C_a, \eta)$
33: if more data needed for distance learning then
34:     goto line 4
35: Learn a policy using an RL algorithm and $W$
```

during the last updated from the constraints, it is possible that more data is needed for W_a to converge. In this paper we instead run the algorithm with different numbers of data collection episodes to show how gathering additional data improves the estimate of W_a and, therefore, the speed of learning (line 35).

In general, collecting data from the environment can be interleaved with distance metric learning and with learning an action-value function. Algorithm 1 simplifies this approach. Rather than updating the distance metric on every time step, it is updated at the end of every episode. This is primarily an implementation detail to reduce the number of times the distance metric learning code (implemented in MATLAB) was called by the simulator (implemented in C).

3.2 Transition Similarity

Algorithm 1 reasons about pairs of vectors, where these vectors describe transitions in the state space: $s \rightarrow s'$. Algorithm 2 calculates the similarity of two vectors, given the current distance metric, where the relatedness of two vectors is at most 1.0 (if they are identical in direction and magnitude). This similarity will be used in the next section to calculate the distance metric under the assumption that states that have similar transitions (for the same action) should be closer in the state space than states that have dissimilar transitions.

Algorithm 2 CALCRELATEDNESS (W, x, y)

```
$\|x\| \leftarrow \sqrt{x^T W x}$
$\|y\| \leftarrow \sqrt{y^T W y}$
$m \leftarrow \frac{\mathit{min}(\|x\|, \|y\|)}{\mathit{max}(\|x\|, \|y\|)}$
$c = \frac{x^T W y}{\|x\| \|y\|}$
return $c \cdot m$
```

Algorithm 3 HOLLER(W, C, η)

```
for each constraint $\langle v, w, x \rangle \in C$ do
    $W_{next} \leftarrow$ minimum over all $W_{next}$ of:
        $D_{\ell d}(W_{next}, W) + \eta \cdot \max(d_{W_{next}}(v, w) - d_{W_{next}}(v, x) + 1, 0)$
    $W \leftarrow W_{next}$
```

3.3 The HOLLER Algorithm

HOLLER (Hinge loss Online Logdet LEarner for Relative distances), as presented in Algorithm 3, is used to learn a distance metric d_W from a list of constraints C and a learning rate η. Recall that each constraint $\langle v, w, x \rangle$ indicates that v should be closer to w than v is to x. The metric learning algorithm follows a standard online updating scheme: each constraint is visited once and the metric is updated after seeing each constraint. As in most online algorithms, we trade off conservativeness with correctiveness when updating the metric. That is, we balance 1) keeping the metric from changing too much from update to update, with 2) updating the metric to satisfy the constraint. This tradeoff is controlled by the learning rate η, and each update to the metric solves an optimization problem that encodes this balance appropriately.

More specifically, we aim to learn a Mahalanobis distance function, which is parameterized by a positive semi-definite matrix W, and is given by $d_W(v, w) = (v - w)^T W (v - w)$. Learning the distance function corresponds to learning the matrix W. Note that since W is positive semi-definite, $W = G^T G$ for some matrix G, and it is straightforward to show that the Mahalanobis distance function d_W is simply the squared Euclidean distance after applying the transformation G to the data points. When updating W to W_{next}, we measure our conservativeness using the LogDet divergence,

$$D_{\ell d}(W_{next}, W) = \text{tr}(W_{next} W^{-1}) - \log \det(W_{next} W^{-1}) - n,$$

where tr refers to the matrix trace and n is the number of rows or columns of W. This divergence measure is natural since positive semi-definiteness of W is automatically maintained, and it has several properties such as scale-invariance which are desirable for metric learning problems. Further, the LogDet divergence has been used extensively in the context of metric learning (e.g., [4, 6]). For correctiveness, we attempt to enforce the constraint $d_W(v, w) \leq d_W(v, x) - 1$, or equivalently, $d_W(v, w) - d_W(v, x) + 1 \leq 0$, as is standard for relative-distance metric learning algorithms [19]. This constraint ensures that the distance between v and w should be much smaller than the distance between v and x. Given these two components, we attempt to find the updated distance parameterized by W_{next} that minimizes the sum of the LogDet divergence between W_{next} and W (conservativeness) plus the error of W_{next} not satisfying the current constraint using the hinge loss (correctiveness), where the sum is balanced by the learning rate η. In particular, we look for a matrix W_{next} that minimizes

$$D_{\ell d}(W_{next}, W) + \eta \cdot \ell(d_{W_{next}}, v, w, x), \qquad (1)$$

where $\ell(d_W, v, w, x) = \max((d_W(v, w) - d_W(v, x) + 1, 0)$ is the *hinge loss* for the constraint $d_W(v,w) \leq d_W(v,x) - 1$. The solution of the minimization problem to compute W_{next} can be computed in closed-form in a manner similar to the online metric learning algorithm of [6]. In particular, a pleasant and surprising aspect of the update for our algorithm is that the solution to W_{next} can be computed as a rank-two update to the matrix W; this can be shown by taking the gradient of (1), setting it to zero, and solving for W_{next}. Details of the update can be found in our publicly available MATLAB code, which show how to handle the gradient at the "hinge" location.[1]

One key advantage of the above online algorithm is that one can prove online regret bounds for this algorithm with appropriate learning rate selection that guarantee that the metric produced by the online algorithm performs similarly to the output of the best possible offline metric learning algorithm (i.e., an algorithm that performs updates of the metric in a batch mode using all constraints). Briefly, one defines the total loss of an online algorithm as the sum of the losses over all T timesteps/constraints. Denote the sequence of W matrices constructed by the online algorithm as $W_1, ..., W_T$, and similarly denote the sequence of v, w, and x vectors from each constraint as $v_1, .., v_T$, $w_1, .., w_T$, and $x_1, ..., x_T$. Then we can define the total loss as

$$\sum_{t=1}^{T} \ell(W_t, v_t, w_t, x_t).$$

Analyses of online learning algorithms focus on the *regret*, which is the difference between the total loss of the online learning algorithm with the total loss of the best possible offline algorithm:

$$Reg = \sum_{t=1}^{T} \ell(W_t, v_t, w_t, x_t) - \text{argmin}_{W_*} \sum_{t=1}^{T} \ell(W_*, v_t, w_t, x_t).$$

The goal is to bound the regret as a function of T, the total number of constraints processed. Our approach, which combines the hinge loss with a convex regularizer, can be viewed as a special case of the online learning framework discussed in Shalev-Shwartz and Singer [13] (see Section 6, equation 38). In particular, with the appropriate selection of learning rates as discussed in Shalev-Shwartz and Singer, we can achieve regret that is bounded by $O(\sqrt{T})$. Finally, note that, while the proposed algorithm shares similarities to existing methods (c.f., [6, 9]) and has been studied theoretically in the context of a large class of online learning methods, we are not aware of metric learning work based on LogDet conservativeness and the standard hinge loss over relative distance constraints.

4. EMPIRICAL VALIDATION

This section introduces a set of six experiments showcasing the benefits of combining HOLLER with Fitted R-MAX.

4.1 2D Mountain Car Domain

This section introduces our experimental domain, a generalized version of the well-studied mountain car task [14]. Mountain car is particularly appropriate for this work as it is a simple domain with continuous state space and can be easily parameterized to highlight the strengths of HOLLER.

In mountain car, the agent must generalize across continuous state variables in order to drive an underpowered car up a mountain to a goal state. To make the problem more challenging than the original formulation, the agent begins at rest at the bottom of the hill.[2] The reward for each time step is -1. The episode ends, and the agent is reset to the start state, after 500 time steps or if it reaches the goal state.

In practice, one of the most difficult challenges for the agent is to find the goal state the first time. After the goal state has been seen at least once, RL algorithms are typically able to quickly learn to consistently find the goal (albeit with different numbers of steps, which determines reward). Effective exploration and generalization is thus critical for agents to quickly find high-performing policies.

In the standard two dimensional mountain car task, two continuous variables fully describe the agent's state. The horizontal position (x) and velocity ($\dot{x}$) are restricted to the ranges $[-1.2, 0.6]$ and $[-0.07, 0.07]$ respectively. The state variables are automatically scaled (linearly) to $[-1, 1]$, as consistent with past work in this domain [7, 14, 18]. If the agent reaches $x = -1.2$, ($\dot{x}$) is set to zero, simulating an inelastic collision. On every time step the agent selects from three actions, {`Left`, `Neutral`, `Right`}, which change the velocity by -0.001, 0, and 0.001, respectively. Additionally, gravity is simulated by adding $-0.025(\cos(3x))$ to $\dot{x}$, which depends on the local slope of the mountain. The goal states are those where $x \geq 0.5$. Our implementation mimics the publicly available version of this task.[3]

4.2 Experimental Procedure

In order to learn in the 2D Mountain Car Domain, we first tune the Fitted R-MAX learning parameters on the standard 2D task without metric learning, and then tune the HOLLER learning parameters on the standard 2D task. The primary consequence of this approach is that the Fitted R-MAX parameters have not been tuned to take advantage of the state variables after metric learning: results we present are therefore biased against HOLLER. Additionally, neither the Fitted R-MAX nor HOLLER parameters are tuned for the variants of the 2D mountain car problem, enabling a fair comparison on the more complex task variants (discussed in Section 4.3).

1: The Standard 2D Mountain Car task is run where agents use Fitted R-MAX with a variety of parameters. The parameters tuned were *minFraction*, which determines if the agent is allowed to end its nearest neighbor approximation early, *modelBreadth*, which sets how fine a uniform grid is used to generalize the state space, and *resolutionFactor*, which determines the size of the regularly spaced grid used to approximate saved instances. We found that values of *minFraction* = 0.01, *modelBreadth* = 0.03, and *resolutionFactor* = 5 produced high-valued policies with few samples and allowed for very fast experiments (in terms of wall clock time). These parameter settings are similar to those used in past experiments in this domain and are explained in detail elsewhere [7, 17].

In order for HOLLER to learn a distance metric, it must have data recorded from the task. To record this data, we allowed the agent to explore the task (with a fully random policy) for different numbers of episodes. The more episodes used for learning the metric, the more likely it will be accurate. However, the episodes spent collecting data will count against the agent's performance (as discussed further in Section 4.3). After trying 6 different values, we decided to experiment with 1, 5, and 10 episodes of data for HOLLER, affecting Algorithm 1, lines 33 and 34.

2: Given the data collected, HOLLER is then used to learn a distance metric. We experimented with 10 values of η (a parameter for Algorithm 1) from 0.0001–0.5 and found that 0.01 and 0.05 produced the best behavior on the 2D Mountain Car task for 1, 5, and 10 episodes. The performance of 0.01 and 0.05 were not dis-

[1] See `cs.lafayette.edu/~taylorm/MetricLearn`

[2] The mountain car task is typically deterministic: to introduce randomness among trials, the initial position of the car in each trial's start state is perturbed by a random number in $[-0.005, 0.005]$, as was done previously in this domain [17].

[3] See `http://library.rl-community.org/wiki/Mountain_Car_(Java)`

tinguishable, suggesting that HOLLER's performance is not overly dependent on this parameter. Experiments in the following sections use η = 0.05. We also tested four values of *NumPts*, the parameter that determines how many temporally similar states to compare, and found that a value of 10 produced slightly better results than 1, 5, or 20.

3: Although HOLLER is designed to be an on-line algorithm, it can be run multiple times over the same constraints if the data is not immediately discarded (Algorithm 1, line 32). In our experiments we tried iterating over the collected data for 1, 2, 3, 5, and 10 times. For 1, 5, and 10 episodes, iterating over the data twice produced slightly better results than the other parameters, but the differences between the final performance (as measured in the following sections) were small. In our experiments, we run iterate over the collected data twice.

4: Having determined all the necessary parameters, HOLLER can be used to learn a distance metric. Initially we learned a single distance metric per action. However, in the Mountain Car domain, the action outcomes are similar enough that the learned distance metrics for the different actions were indistinguishable. Therefore, the experiments below focus on learning a single distance metric, $W_{Neutral}$ (using only instances where the agent randomly executed the `Neutral` action) and using that metric for all W_a when learning an action-value function.

5: To evaluate HOLLER, we then learn the 2D Mountain Car task using Fitted R-MAX, with and without the learned distance metrics. The effect of the distance metric is compared in the following sections by evaluating the final and total rewards using both the Euclidean distance and using the learned W_a.

4.3 2D Mountain Car Results

First, consider the distance metric, W, learned by HOLLER from 10 episodes worth of data. Examining the 10 trials, we find that

$$W = \begin{bmatrix} 0.119 \pm 0.012 & -0.006 \pm 0.003 \\ -0.006 \pm 0.003 & 0.096 \pm 0.008 \end{bmatrix},$$

where the ± terms show the standard error. The values on the diagonal show that x, the first state variable, is slightly more important than $\dot{x}$, the second state variable. The off-diagonal values are very small, showing that linear combinations of the two state variables are not critical in this domain. However, it is impossible to say whether this distance metric is "correct" – instead, the utility of this metric is in the observed performance of the RL agent.

Figure 1(a) shows learning curves for learning the 2D Mountain Car task with Fitted R-MAX, both with (for 1, 5, or 10 episodes of data) and without (No Metric Learning) HOLLER. The x-axis shows the episode and the y-axis shows the average reward for that episode number. Error bars show the standard error over 10 independent trials. All experiments are averaged over 10 trials and all experiments in this section are ended after 100 episodes. The three trials that use HOLLER after collecting data for 1, 5, and 10 episodes learn to reach the goal very quickly, quickly outperforming learning with the no distance metric. However, this analysis does not account for the number of episodes spent collecting data (Algorithm 1, lines 5–11).

Figure 1(b) explicitly shows the time spent collecting data for HOLLER; for instance, when collecting data for 10 episodes, the learning curve begins on episode 10, as episodes 0-9 are assumed to have reward -500. To make the graph more readable, a 5-episode sliding window is used and error bars are not shown. Additionally, the performance of Sarsa (a popular model-free learning algorithm) with CMAC function approximation is compared by using the same parameters as those in the literature [7, 14, 17]), showing that Sarsa agents take longer to discover the goal state, but that eventually achieve a slightly higher reward.

One reasonable dimension along which to evaluate the effectiveness of HOLLER would be the average reward at a set amount of data (e.g., after 100 episodes). However, such a metric ignores the "speed" of learning — Sarsa has a higher performance at episode 100 but suffers from a slow start. Analyzing the cumulative rewards also shows that using Fitted R-MAX with HOLLER learning from 1 episode of data outperforms the other learning methods.

In the standard 2D Mountain Car problem, HOLLER with 1, 5, and 10 episodes of data outperforms Fitted R-MAX without HOLLER in terms of the final average reward and the cumulative reward. Additionally, the difference in cumulative rewards is statistically significant. While Sarsa outperforms Fitted R-MAX on this test both in terms of final and cumulative reward, previous work has shown that it is difficult for Sarsa to scale to higher-dimensional versions of this problem [18]. Experiments showing the superiority of Fitted R-MAX are replicated later in Section 4.4. A summary of this and other experiments can be found in Table 1.

4.3.1 *Variant 1: Inflated State Variable*

As a second task, we consider the more general case where the range of the second state variable is not known. The state variable $\dot{x}$ still ranges from [-0.007, 0.007], but we assume that in order to ensure that all data is scaled so that all state variable ranges are within the expected range of [-1, 1], $\dot{x}$ is divided by 0.7 (rather than 0.007), causing the observed range to become [-0.01, 0.01]. Such non-optimal scaling could occur if the human designer did not know the true variable range and was being careful. Alternatively, the range could be automatically determined through sampling the minimum and maximum values, but two noisy readings (one high and one low) could throw off the scaling. As seen in the previous subsection, the x and $\dot{x}$ state variables are both important for accurately predicting the transition function and we would expect that Fitted R-MAX, using parameters set for the standard 2D mountain car task, will not perform as well as when it is coupled with a learned distance metric.

As shown in Figure 2(a), the episodes spent learning the distance metric initially hurt the learners: Fitted R-MAX without a distance metric initially outperform an agent that collected 10 episodes of data for HOLLER. However, the final average reward and average cumulative reward is better for all three settings of the HOLLER agents, although the differences are only statistically significant about half of the time (see Table 1 for Student's t-test results).

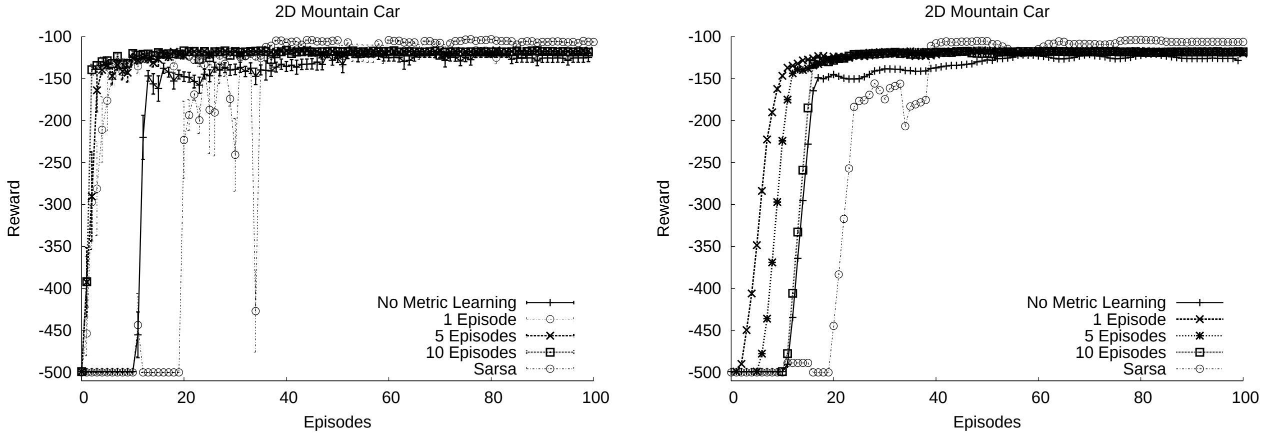
4.3.2 *Variant 2: Sensor and Actuator Noise*

To test the efficacy of HOLLER in the presence of noise, we next consider a variant of mountain car that includes partial observability and stochasticity. As before, the position and velocity state variables are scaled to the range $[-1, 1]$ and then Gaussian noise is added to the agent's observation, drawn randomly on each time step from $\mathcal{N}(0, 0.1)$. Similarly, on every time step, the agent's velocity is multiplied by zero-mean noise drawn from $\mathcal{N}(0, 0.01)$.

Figure 2(b) shows that although the noise makes learning more difficult for all learners (i.e., their reward is lower than agents in Figure 1(b)), HOLLER is able to learn distance metric functions that allow the agents to outperform the default scaling. This is a particularly important test as it shows that HOLLER is robust to noise, as desired. Using HOLLER produces a higher final and cumulative reward in all three cases, although only the differences between the cumulative rewards are statistically significant.

4.3.3 *Variant 3: Irregular Action Function*

Next, consider the situation where the transition function is highly



(a) The episodes used by HOLLER are ignored, with standard error

(b) 2D Normal

Figure 1: These figures show the same learning curve data where the x-axis is the episode number and the y-axis shows the reward. In (a), the y-axis shows the average reward on a given episode (higher is better) with the standard error. (b) also shows the average reward per episode, but accounts for the episodes spent learning the distance metric and uses a 5-episode sliding window.

dependent on the state, as was done in the 2009 Reinforcement Learning Competition (c.f., `http://2009.rl-competition.org/` and [21]). In particular, the actions 0–2 (`Left`, `Neutral`, and `Right`) were mapped such that the action executed by the agent depended on $\dot{x}$ and a (the action selected by the agent). The action executed by the car in the simulator was

$$\left(a+\left(\frac{\dot{x}+0.07}{0.14}\cdot 99.0\right)\right) \mod 3.$$

As expected, Figure 3(a) shows that learning a metric significantly improves learning, both in terms of the final reward and cumulative reward, as the learned metric can automatically increase the resolution to $\dot{x}$, allowing it to better approximate a transition function significantly more complex than for the standard 2D mountain car.

4.3.4 Variant 4: A Third, Irrelevant, State Variable

As a final variant for the 2D Mountain Car task, we consider adding an additional irrelevant state variable. Although the transition and reward functions still depend only on x and $\dot{x}$, the agent is provided a random number as a third feature on every time step. This state variable is drawn uniformly in [-0.025, 0.025]. As Figure 3(b) shows, this additional state variable significantly degrades the performance of Fitted R-MAX with a Euclidean distance metric as it must now generalize its data over an extra dimension (i.e., it suffers from the"curse of dimensionality"). However, HOLLER allows this third state variable to be de-valued, allowing the agents learn almost as well as in the standard 2D mountain car task.

HOLLER is not dependent on the number of state variables: although Fitted R-MAX can generally not scale to high-dimensional spaces, using HOLLER would allow an experimenter to eliminate irrelevant state variables, potentially enabling this and other methods to scale to much higher dimensional spaces.

4.4 4D Mountain Car

The 4D Mountain Car task extends the 2D task so that there are four state variables $(x, \dot{x}, y, \dot{y})$ and the agent selects from five actions (`Neutral`, `West`, `East`, `South`, `North`) [18]. The transition function is similar to the 2D case, but now takes into account the extra dimensions. Likewise, the goal region is now $x \geq 0.5$ and $y \geq 0.5$. Our task implementation is based on a publicly available implementation.[4] This task is much more difficult than the 2D task because of the increased state space size and additional actions. After initial experimentation without distance metric learning, we set the parameters of Fitted R-MAX to be similar to past work [17] *minFraction* = 0.3, *modelBreadth* = 0.3, *resolutionFactor* = 3, and agents train for a total of 250 episodes.

As shown in Figure 3(c), the final and cumulative performance of learners using HOLLER is higher than those that rely on the Euclidean distance metric. Also, note that Sarsa, using the same parameters set in the literature [18], does much wore than Fitted R-MAX, due to the high-dimensional space. Sarsa agents do not consistently find the goal state until after 2,000 episodes, requiring roughly two orders or magnitude more data than the instance-based learning method (with or without metric learning).

Taken as a whole, and summarized in Table 1, these experiments show that HOLLER can successfully improve learning performance on a variety of tasks, both in terms of final and cumulative reward.

5. RELATED WORK

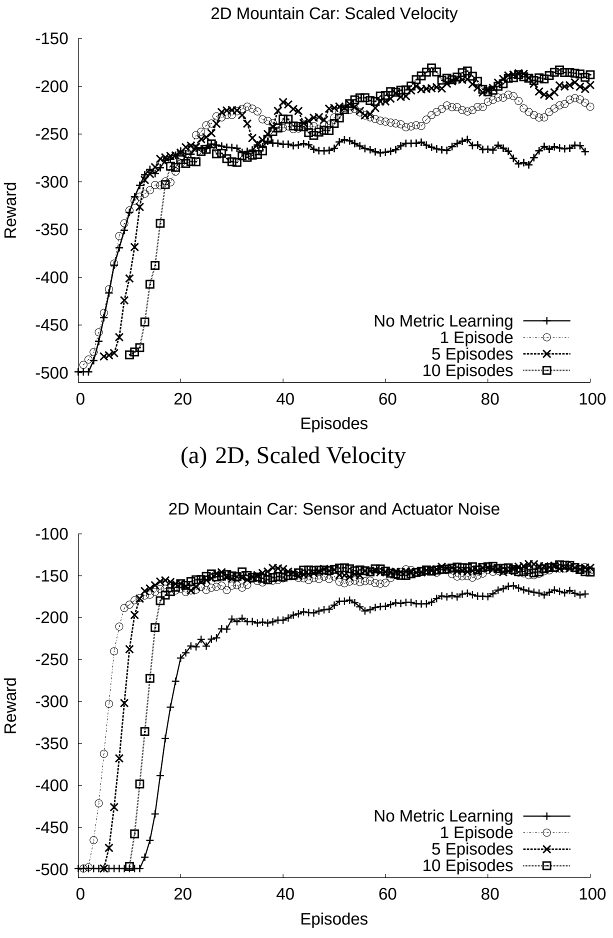
The most similar distance metric learning work has been discussed earlier in Sections 2.3 and 3.3. This section focuses on the most relevant existing reinforcement learning algorithms.

Graph-based approaches to learning state representations, such as using *proto-value functions* [10], typically focus on using a known connectivity graph (e.g., a transition function) to learn a (near-) optimal set of features. By using the eigenvectors of the connectivity graph's Laplacian, very accurate representations of an MDP's value function can be learned. However, proto-value function work does not typically consider the sample complexity of learning such a connectivity graph — our work is directly concerned with minimizing the amount of environmental samples needed to learn a state representation and thus attempt to maximize the on-line reward.

The Bellman Error Basis Functions (BEBF) [12] method relies on iteratively adding basis functions, where each basis function is constructed to improve the Bellman error over the previous set of basis functions. BEBF differs from the current work primarily in its aim — while the BEBF work examines relatively simple RL tasks with the goal of constructing very accurate value functions from hundreds of thousands of samples, HOLLER instead aims to construct a distance metric with relatively little data that can be used to both guide exploration and improve value function estimation.

In a supervised learning setting, unlike in RL, training sets provide the correct target label, enabling a more straightforward appli-

[4] `http://library.rl-community.org/wiki/Mountain_Car_3D_(CPP)`



(a) 2D, Scaled Velocity

(b) 2D, Sensor and Actuator Noise

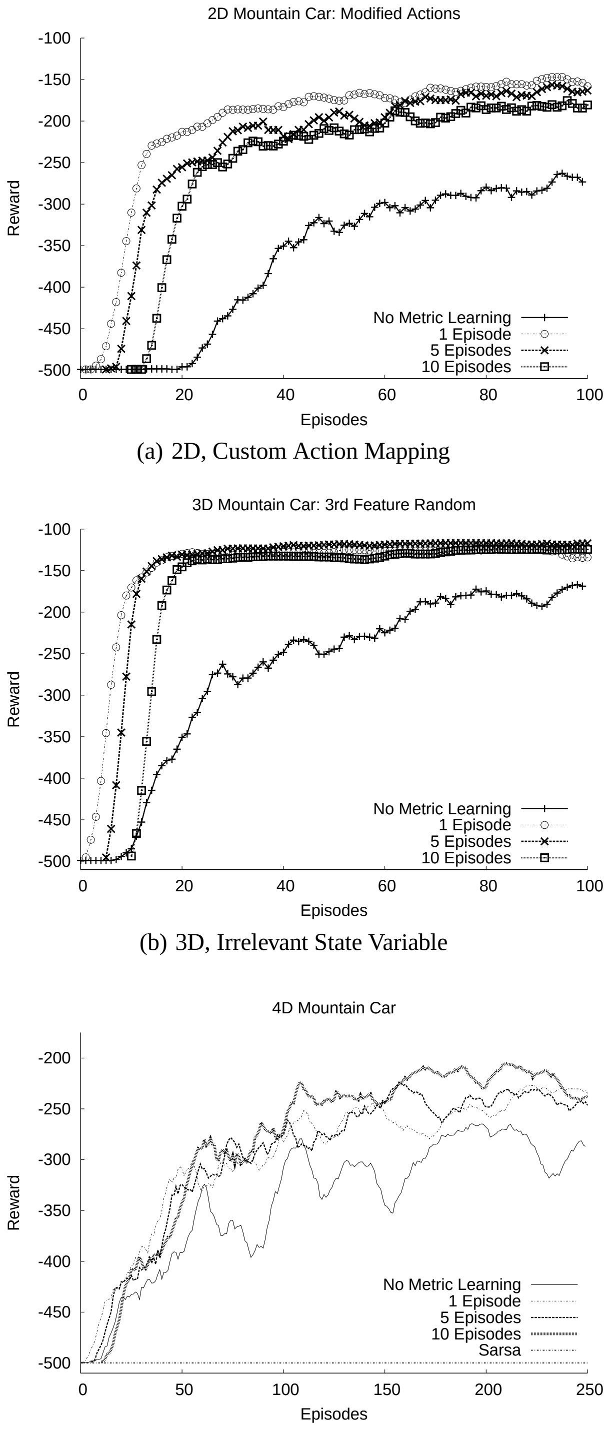
Figure 2: A learned distance metric improves both the total and final reward when the velocity state variable is incorrectly scaled (a) and when there is noise in both the sensors and actuators (b).

cation of distance metric learning. For instance, *Metric Learning for Kernel Regression* [20] (MLKR) is a metric learning method designed for regression problems.

Three recent papers presented at ECML-10 also tackle similar problems. Nouri and Littman [11] build upon MLKR to create the *Dimension Reduction in Exploration* algorithm. The algorithm constructs a set of "factorized" MLKR problems (F-MLKR), under the assumption that individual state features for resulting states are independent of each other, where one MLKR problem is constructed per state feature, per action, for a total of $||A|| \times ||S||$ F-MLKR regressors. F-MLKR agents must also be provided the reward function, unlike in HOLLER, where the reward is learned. Additionally, agents that use HOLLER benefit from dimensionally reduction as well as proper scaling of state variables, and can be combined with existing RL methods.

The second recent paper, Jung and Stone [8], trains multiple Gaussian processes in batches to approximate the transition function. The GP-RMAX algorithm requires a deterministic transition function, must be provided the reward function. In contrast to both F-MLKR and GP-RMAX, HOLLER learns a distance function for the entire state space based on few samples, which means that HOLLER can quickly generalize over the entire state space.

The third paper [2] presents an actor-critic method to determine where to place basis functions and what parameterization they should



(a) 2D, Custom Action Mapping

(b) 3D, Irrelevant State Variable

(c) 4D Mountain Car: Performance

Figure 3: Figures (a) and (b) show how HOLLER produces better learning in task with a custom action mapping and with an irrelevant state variable, respectively. In (c), learning curves are averaged over ten trials with a 10-episodes sliding window.

have, rather than learning a single metric that is useful across the state space (independent of the function approximator parameterization). Additionally, we note that the authors test their algorithm on an easier version of mountain car (where the agent starts at a random state rather than the bottom of the hill, making exploration sig-

Domain	Algorithm	Final Ave. Reward	Stat. Sig.	Cumulative Reward	Stat. Sig.
2D: Standard	Fitted R-MAX	-126		-17600	
	HOLLER-1	-118		-13620	✓
	HOLLER-5	-118		-14783	✓
	HOLLER-10	-117		-16440	✓
	Sarsa	-106	✓	-19755	(✓)
2D: Scaled	Fitted R-MAX	-268		-28050	
	HOLLER-1	-227		-25380	
	HOLLER-5	-199	✓	–24740	✓
	HOLLER-10	-199	✓	–26000	
2D: Noisy	Fitted R-MAX	-157		-23600	
	HOLLER-1	-136		-16840	✓
	HOLLER-5	-141		-17240	✓
	HOLLER-10	–150		-18733	✓
2D: Convoluted Actions	Fitted R-MAX	-260		-36190	
	HOLLER-1	-154	✓	-19990	✓
	HOLLER-5	-161	✓	–22660	✓
	HOLLER-10	-177	✓	-25460	✓
3D: Irrelevant Feature	Fitted R-MAX	-164		-26360	
	HOLLER-1	-128	✓	–14500	✓
	HOLLER-5	-117	✓	-14840	✓
	HOLLER-10	-124	✓	-17630	✓
4D: Standard	Fitted R-MAX	-291		-36190	
	HOLLER-1	-225		-19990	✓
	HOLLER-5	-239		-22663	✓
	HOLLER-10	-241		-25460	✓
	Sarsa	-500	(✓)	-50000	(✓)

Table 1: This table summarizes all experiments, averaging over ten independent trials. The third column shows the average reward at the end of the trial (250 episodes for the 4D task, 100 episodes for all others). The fourth column has a check if the difference in the final reward is statistically significantly different from learning with Fitted R-MAX without a learned distance metric, as determined by $p < 0.05$ on Student's t-test results. The fifth and sixth columns report the average cumulative reward and whether the difference in the cumulative rewards and Fitted R-MAX are statistically significant.

nificantly easier), but their algorithm takes thousands of episodes to converge.

6. CONCLUSION AND FUTURE WORK

This paper has introduced HOLLER and shown how it can be combined with an off-the-shelf instance based RL algorithm. Empirically, this novel distance metric learning algorithm significantly improves learning efficacy in a number of different tasks, including noise and irrelevant state variables. One of the key benefits of HOLLER is that very little data is required to learn an appropriate state representation and thus the on-line reward can be significantly improved relative to learning with a Euclidean distance metric.

In the future, we intend to try to fully integrate learning W and a control policy simultaneously. While such an integration would not be critical in domains where the distance metric can be quickly learned, it may prove useful in more complex and higher-dimensional tasks. We also are interested in attempting to further improving the efficacy of HOLLER by trying establish appropriate decay rates for η (rather than using a fixed learning rate), combining the updates from multiple actions (rather than learning each W_a in isolation), and trying to tune exploration to learn W as quickly as possible (rather than relying on random exploration). Lastly, while this paper has focused on Fitted R-MAX, we expect that HOLLER would be beneficial to other instance-based RL methods, as well as model-free methods. For instance, future work could examine how W could be used by Sarsa to help select, or parameterize, its function approximator so that the value function can better match the underlying topology of the state space without relying on human intuition or simple estimates of state variable ranges. Lastly, it would be interesting to empirically compare our Mahalanobis distance approach, with the LogDet loss function, to alternative approaches.

Acknowledgements

The authors would like to the anonymous reviewers and Tobias Jung for useful comments and suggestions.

7. REFERENCES

[1] J. S. Albus. *Brains, Behavior, and Robotics*. Byte Books, Peterborough, NH, 1981.

[2] D. D. Castro and S. Mannor. Adaptive bases for reinforcement learning. In *ECML*, 2010.

[3] J. Davis and I. Dhillon. Structured metric learning for high-dimensional problems. In *KDD*, 2008.

[4] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon. Information-theoretic metric learning. In *ICML*, 2007.

[5] A. Globerson and S. Roweis. Metric learning by collapsing classes. In *NIPS*, 2005.

[6] P. Jain, B. Kulis, I. Dhillon, and K. Grauman. Online metric learning and fast similarity search. In *NIPS*, 2008.

[7] N. K. Jong and P. Stone. Model-based Function Approximation for Reinforcement Learning. In *AAMAS*, 2007.

[8] T. Jung and P. Stone. Gaussian processes for sample efficient reinforcement learning with RMAX-like exploration. In *ECML*, 2010.

[9] B. Kulis and P. Bartlett. Implicit online learning. In *ICML*, 2010.

[10] S. Mahadevan and M. Maggioni. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

[11] A. Nouri and M. L. Littman. Dimension reduction and its application to model-based exploration in continuous spaces. In *ECML PKDD*, 2010.

[12] R. Parr, C. Painter-Wakefield, L. Li, and M. L. Littman. Analyzing feature generation for value-function approximation. In *ICML*, 2007.

[13] S. Shalev-Shwartz and Y. Singer. A primal-dual perspective of online learning algorithms. *Machine Learning Journal*, 2(69):115–142, 2007.

[14] S. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[15] M. Slaney, K. Weinberger, and W. White. Learning a metric for music similarity. In *ISMIR*, 2008.

[16] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[17] M. E. Taylor, N. K. Jong, and P. Stone. Transferring instances for model-based reinforcement learning. In *ECML PKDD*, 2008.

[18] M. E. Taylor, G. Kuhlmann, and P. Stone. Autonomous transfer for reinforcement learning. In *AAMAS*, 2008.

[19] K. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, 2006.

[20] K. Q. Weinberger and G. Tesauro. Metric learning for kernel regression. In *AI-STATS*, 2007.

[21] S. Whiteson, B. Tanner, and A. White. The reinforcement learning competitions. *AI Magazine*, 31(2):81–94, 2010.

[22] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *NIPS*, 2002.

[23] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, 2003.