Explicit Manifold Representations for Value-Function Approximation in Reinforcement Learning

William D. Smart

Department of Computer Science and Engineering
Washington University in St. Louis
One Brookings Drive
St. Louis, MO 63130
United States
wds@cse.wustl.edu

1 Introduction

We are interested in using reinforcement learning for large, real-world control problems. In particular, we are interested in problems with continuous, multi-dimensional state spaces, in which traditional reinforcement learning approached perform poorly.

Value-function approximation addresses some of the problems of traditional algorithms (for example, continuous state spaces), and has been shown to be successful in a number of specific applications. However, it has been proven not to work in the general case, and has been known to fail even on simple problems.

We propose a novel approach to value-function approximation, based on the theory of manifolds. We identify the key failing of current techniques, and show how our approach avoids this problem by explicitly modeling the topology of the state space.

We begin with a brief description of the problems of current value-function approximation techniques. We then motivate our proposed approach, outlining the mathematics underpinning it, and provide the results of our initial experiments.

2 The Problem with Value-Function Approximation

Value-function approximation (VFA) is a technique in reinforcement learning (RL) where the tabular representation of the value function is replaced with a general-purpose function approximator. It is generally used to deal with continuous state spaces, and to allow generalization between similar states.

The straightforward application of VFA, where the tabular representation is simply replaced by a general-purpose function approximator, has been shown not to work in the general case [2]. However, in certain simple worlds, it seems to work quite reliably, even when Gordon's conditions [3] are not fully satisfied. For example, consider the world shown in figure 1. The agent can move in the four cardinal directions, and starts in the lower left corner. Rewards are -1 for hitting a wall, +1 for reaching the upper right corner, and zero everywhere else. The agent's state corresponds to its position in the world, and is represented by a vector in $\mathbb{R}^2$. The policy shown by the blue line was learned using Q-learning [10] with a two-layer back-propagation network as a value-function approximator. The agent reliably learned an optimal policy for this world from a variety of different random starting configurations.

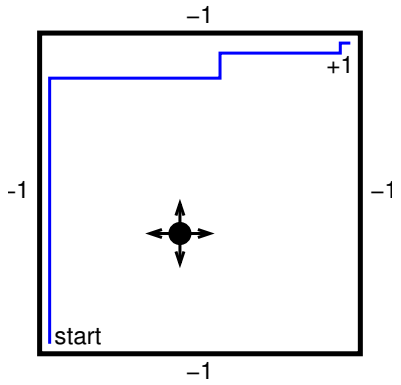


Figure 1: A simple world where straightforward VFA is effective, with a policy learned using Q-learning with VFA (blue line).

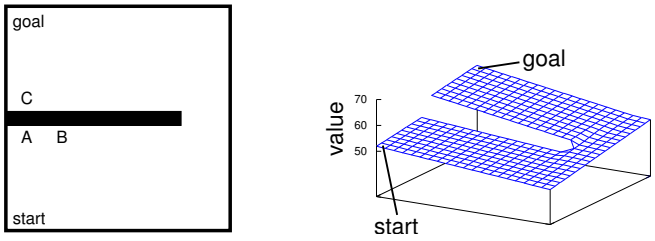


Figure 2: A simple world where VFA fails (left). The value function for the "up" action, calculated by tabular Q-learning (right).

Although using value-function approximation in the above example works well, we can cause it to fail by simply adding a wall (see figure 2). Typically, the learned policy will seem to ignore the wall, and attempt to drive through it towards the goal. The reason for this failure is easy to understand. VFA works by generalizing values from known states to unknown ones. There is an implicit assumption that states close to each other have similar values. For all commonly used function approximators, this measure of similarity is the (possibly weighted) Euclidean distance between the states. For the world shown in figure 2, this assumption is flawed in some parts of the state space. In particular, points A and C are close in Euclidean space, but have very different values. This difference represents itself as a discontinuity in the value function. VFA with a simple function approximator will tend to "blur" this discontinuity, attempting to incorrectly generalize across it.

The main reason for the failure of VFA is that most function approximation algorithms implicitly make assumptions about the topology of their domain, which are often not met in practice. To separate the points, and cause VFA to succeed, we need to correctly model the topology of the domain of the VFA algorithm. We propose doing this by explicitly modeling the domain using a *manifold*. We give a more formal definition of a manifold below. First, however, we give a simple example to build the intuition of a domain with a different topology.

Consider the simple world shown on the left of figure 3, an open room with an obstacle in the middle. As in figure 2, the Euclidean distance (red lines) between points A and B, and between A and C is small. However, B is intuitively "closer" to A than C is.

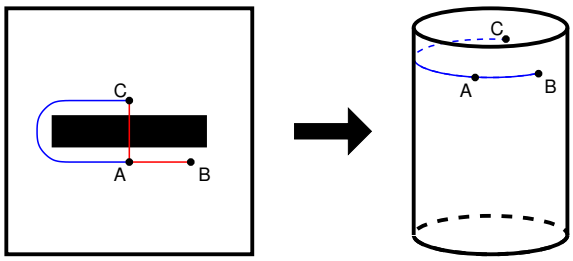


Figure 3: A simple world containing one obstacle (left). The world is topologically cylindrical (right).

The state space of the world is a closed subset of $\mathbb{R}^2$, corresponding to the position of the agent. A closed subset of $\mathbb{R}^2$ with a single continuous boundary is topologically a disk. However, the manifold on which the agent can actually move (the free space) is topologically a cylinder, as can be seen in figure 3. The top edge of the cylinder corresponds to the outer edge of the obstacle, and the bottom edge to the outer wall of the world. We can safely use Euclidean distance on the surface of the cylinder to get a better measure of "nearness". The distance on the cylinder from A to C (blue line) roughly corresponds to the blue line on the left side of the figure.

The Euclidean distance metric on the cylinder corresponds to a step-based distance metric in the original world. Intuitively, this makes sense for VFA, since the value function is defined in terms of the number of steps to reward-giving states, $V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$. The value of a state will now be similar to those that are close to it in the manifold.

3 Manifolds

Before giving the formal definition of a manifold, we provide an intuition about the structure, and how it can be used constructively. Consider an atlas of the world. Each page in the atlas has a partial map on it, usually a single country. Each of these single-page maps can have a different scale, depending on the size of the country. This allows more detail to be shown where it is necessary, without committing to representing the entire atlas at a particular resolution.

The pages in the atlas also overlap with each other at the edges. For example, the map for France has part of northern Spain on it. This provides a well-defined way of moving from one map page to another. Notice that the maps may be at different scales, and may not line up perfectly. page to another, we can establish a correspondence between points in the overlap regions.

This method of using overlapping partial maps allows us to map a complex surface (the world) with a set of simpler surfaces (the individual maps). Each local map can be appropriate to the local features, such as scale, and be topologically simpler than the global map. In the case of an atlas, we are covering a sphere with a set of disks. We can define global features, such as distance, by composing their local versions on each page, and dealing appropriately with the overlap regions. This will prove to be important to us, and we will discuss it in more detail below.

We now make some definitions. Each map page in the above example is a *chart*. The collection of all charts is the *atlas*. The area shared by two adjacent pages is the *overlap region*. The function that maps points in the France map to points in the Spain map is the *transition function*. The function on each chart (for example, the elevation, or value function) is the *embedding function*.

We formally define a manifold, $\mathcal{M}$, embedded in a space, $\mathcal{S}$, as an atlas, a set of overlap regions and a set of transition functions.

Atlas A finite set, A, of homeomorphisms from $\mathcal{S}$ to the disk. Each element $\alpha_c \in A$ is called a *chart*. The co-domain of α_c is labeled as c. The term *chart* is commonly also used to refer to c.

Overlap Regions A set of subsets, $U_{ij} = c_i \cap c_j$, where α_{c_i} and α_{c_j} are charts in A and where $U_{ii} = c_i$. U_{ij} will be empty if c_i and c_j do not overlap. The atlas, A, defines U_{ij} in the following way: A point $p \in c_i$ is in U_{ij} if and only if there exists a point $q \in c_j$ such that $\alpha_{c_i}^{-1}(p) = \alpha_{c_j}^{-1}(q)$.

Transition Functions A set of functions, Ψ. A transition function $\psi_{ij} \in \Psi$ is a map $\psi : U_{ij} \rightarrow U_{ji}$ where $U_{ij} \subset c_i$ and $U_{ji} \subset c_j$. Note that U_{ij} and U_{ji} will often be empty. We can define ψ_{ij} to be $\alpha_i \circ \alpha_j^{-1}$.

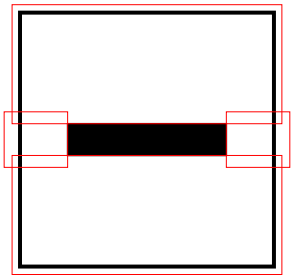


Figure 4: One possible chart allocation for a simple world with one obstacle.

Most of the previous work with manifolds involves using manifolds to analyze an existing surface. We propose constructing a manifold to model the domain of the value function. Some papers in the computer graphics literature describe the use of manifolds for surface construction and modeling [4, 5, 6, 8, 9]. However, they all deal with two-dimensional surfaces embedded in three-dimensions.

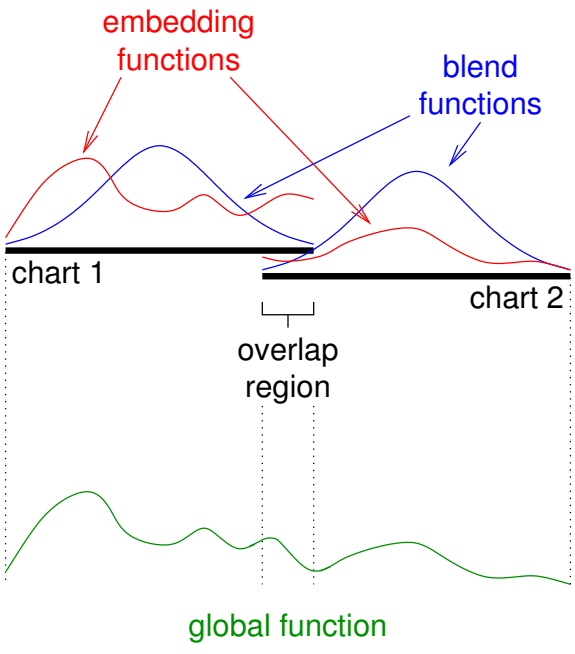


Figure 5: Blending across charts.

We are, in general, interested in manifolds of dimension n embedded in m-dimensional spaces, where $n \leq m$. For the remainder of this proposal we will, without loss of generality, only deal with the case where $n = m$. This makes the mapping, $\alpha_c : \mathcal{S} \rightarrow c$, and the corresponding transition functions simple and intuitive.

Now that we have defined what we mean by a manifold, the first task is to use the manifold to model the domain of the value function. The basic idea is to model the reachable parts of the state space as a set of simpler regions (the charts). Consider the world shown in figure 4. As we established previously (on page 3), this world is topologically cylindrical. There are an infinite number of possible atlases for this example; any set of charts that completely covers the reachable part of the state space is acceptable. One possible atlas is shown in figure 4. Notice that charts on opposite sides of the obstacle do not overlap. The reachable space in each chart is convex; there are no obstacles

in any of the charts.[1] Euclidean distance should, therefore, be a reasonable measure of similarity within a chart. This means that straightforward VFA techniques should work well with in the chart.

Now we have a model of the domain of the value function. The next step is to embed the value function approximation in this domain. To create the value function, we build a local value function on each chart, and combine the results. The values of the embedding functions (the local value functions) are combined as shown in figure 5. In this figure, there are two one-dimensional charts, each with an embedding function (the red line). Each chart also has a *blend function* defined on it. This is a smooth, continuous function, $b_c : c \rightarrow \mathbb{R}$. The value of the blend function, and all of its derivatives, must be zero at the edge of the chart. The blend functions form a partition of unity over the manifold. Embedding functions are combined linearly in the overlap regions, according to the blend function. The global function, F, at point p is defined as a linear blend of the embedding functions, f_c, of all the charts in which p lies, weighted by the blend functions,

$$F(p) = \frac{\sum_{c \in A} b_c(p) f_c(p)}{\sum_{c \in A} b_c(p)}.$$

Recall that, for most values of p, most values of $b_c(p)$ will be zero, since $p \notin c$.

Following Grimm and Hughes [4], we use a one-dimensional spline as the blend function, $b(x)$. If we assume a chart covering the unit square, $b(0) = 0$, $b(1) = 0$, and $b(x)$ is maximal at $x = 0.5$. All derivatives are zero at the boundaries, $b^{(n)}(0) = 0$, $b^{(n)}(1) = 0$. To calculate the blend value for a particular point in the chart, $p \in \mathbb{R}^n$, the blend function is applied to each dimension, and the value multiplied together,

$$b_c(p) = \begin{cases} \prod_{i=1}^{n} b(p_i), & p \in c, \\ 0, & \text{otherwise.} \end{cases}$$

Blending the embedding functions in this manner allows us to make some guarantees about the resulting global function. Building a global function from this manifold, under certain technical conditions, will result in a C^k (smooth and continuous up to the kth derivative) manifold surface. Also, the value of the global function will be locally bounded by the values of the embedding functions, since the blend functions are a partition of unity.

3.1 Manifolds for VFA

The basic idea of using manifolds for VFA is to allocate a set of overlapping charts that cover the world, reflecting the topology of the reachable space. We assign a function approximator to each of these charts, and construct the global value function by blending the predictions of these function approximators together, as described above. If we are using Q-learning, we construct one manifold for each possible action. Updating the value of a state corresponds to presenting a training example to the function approximator for each of the charts in which the state lies. In all other ways, we can treat the set of manifolds as a drop-in replacement for the table in standard RL algorithms. However, this somewhat simplistic description hides two hard problems: how to allocate charts, and how to calculate distance in the manifold. We are currently investigating computational approaches to these problems.

[1] Formally, each chart is topologically a disk.

In this paper, we will consider only approximations of the state-action value function, which can be calculated off-policy. The state-value function, on the other hand, is calculated for a particular policy. Thus, a change in the policy will affect which areas in the environment are reachable. This, in turn, will affect the effective topology of the world. This means that the best chart allocation will depend on the (possibly changing) policy. We will also simplify our initial experiments by assuming that there is a single source of reward at the goal state.

We also make the assumption in this paper that we can arbitrarily sample states and actions, and that we are not contstained to follow trajectories. Once we have a better understanding of chart allocation schemes in this setting, we hope to relax the the assumption and deal with trajectory-based agents. It seems possible, in principle, to construct charts on-line, as the agent follows trajectories through the world.

4 Preliminary Empirical Results

We have performed some preliminary experiments to investigate how well manifold-based representations of the value function work in practice. We have experimented with two very straightforward chart allocation schemes: fixed-size random placement, and random-walk based.

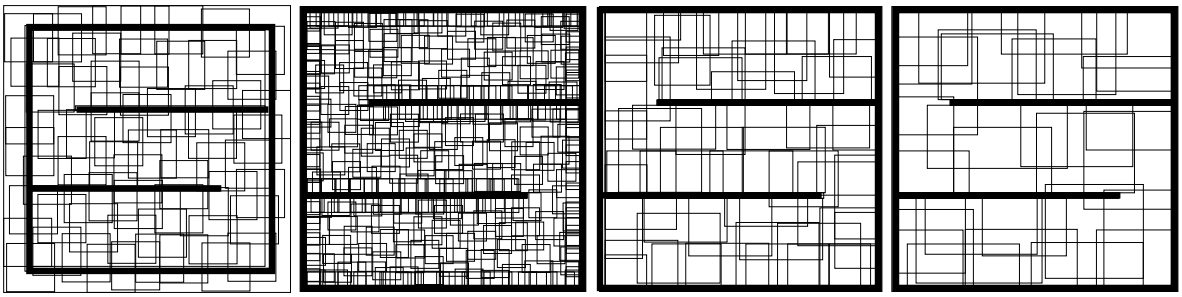
Fixed-Size Random: Pick a random point not currently covered by the manifold. Place a fixed-size, axis-aligned square chart at that point. Repeat until the domain is completely covered by charts.

Random Walk: Pick a random point not currently covered by the manifold as the starting point. Perform n random walks of length m steps from that point by randomly selecting actions. Record all of the points visited. Place the smallest axis-aligned rectangular chart that covers all of these points. Repeat until the domain is completely covered by charts.

The random walk allocation scheme is the simplest "realistic" allocation scheme. The intuition is that points a small number steps away from each other are likely to be similar, and should be in the same chart. Ideally, we would like to explore all possible m-step paths from the starting point, but this becomes intractable for a moderate number of actions and size of m. Thus, we sample using a random walk, and trust this to give us reasonable coverage.

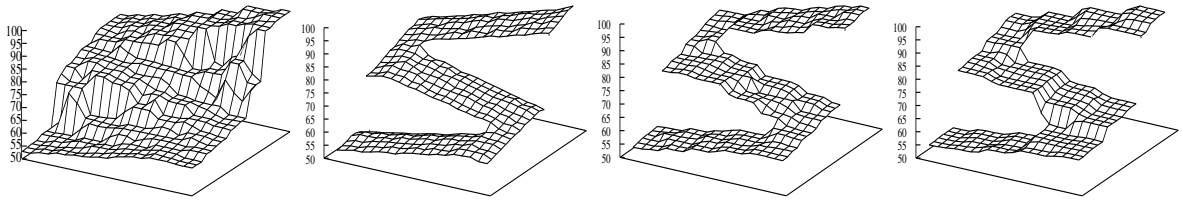
Finding a point not covered by the manifold is a hard problem, and we do not currently have a satisfying solution. It is easy to tell if a point is covered by checking each of the charts in the manifold. We use a generate-and-test approach, randomly picking points and checking if they are covered. We keep picking points until we come up with one that is not covered. If we cannot generate such a point in a fixed number of tries (one million for these experiments), we declare that the domain is completely covered by the manifold.

Figure 6 shows the various chart allocation strategies. Notice that the fixed-size allocation overlaps the internal walls in the world, and does not model the topology. The random walk allocations do not straddle the walls, and do model the topology of the domain. Figure 7 shows the corresponding learned value functions for the "up" action, for each of the allocations. The discount factor was set to 0.99, the learning rate to 0.8, and one million random training experiences were used. A simple leaky-integrator (constant) function approximator was used as the embedding function for each chart. Notice that the fixed-sized allocation (figure 7(a)) does not capture the



(a) (b) (c) (d)

Figure 6: Chart allocations for (a) fixed-size charts, and random walks of (b) length 1, (c) length 3, and (d) length 5.



(a) (b) (c) (d)

Figure 7: Learned value functions for (a) fixed-size charts, and charts allocated by random walks of (b) length 1, (c) length 3, and (d) length 5.

discontinuities in the value function well. However, each of the random-walk based allocations capture the discontinuity well.

With small charts, as in figure 6(b), all points covered by a chart are very similar, and the value function approximation is very smooth. As the charts become larger, the approximation begins to reflect the underlying chart structure, as in figure 6(d). This is especially true if we use a simple embedding function, as we are here. There is a fundamental tension between the complexity of the embedding function approximator and the maximum useful chart size. Expressive function approximators permit larger charts because they can model the value of points that are relatively dissimilar. Simple function approximators necessitate small charts, since they can only model the value of points that are very similar.

5 Conclusions and Current Work

Manifold representations offer the possibility of uniting a number of other successful value function approximation schemes. For example, a tabular representation is simply a manifold with uniformly-sized charts that tile the state space, and have zero overlap, and a leaky-integrator function approximator as the embedding function. Variable-resolution partitioning schemes [7] can be viewed as cahrt-allocation, again with non-overlapping charts. Interestingly, one of the most successful function approximators for VFA, the CMAC [1] is exactly a manifold. A CMAC tiles the state space with overlapping tiles, each of which has a constant value assigned to it. The prediction for a point, p, is the sum of the values of all of the tiles in which p falls. This sum is often weighted

by the distance of p from the center of the tile. Using the language of manifolds, the tiles are charts, the value is a leaky-integrator embedding function, and the weights are the blend functions.

For non-overlapping charts, our use of manifolds bears some similarity to piecewise function approximation schemes. For example, for a leaky-integrator embedding function and non-overlapping charts, a manifold is exactly a piecewise constant function approximator. The addition of overlap regions causes the approximation to become smooth and continuous over the state space. The main difference between our method and existing similar function approximators is that we explictly attempt to model the topology of the state space.

Although our experiments to date have yielded promising results, it remains an open question as to whether constructing an effective manifold is fundamentally easier than solving the problem through other methods, or learning models of the transition and reward functions. To address this question empirically, we are currently working on chart-allocation schemes based on statistical tests. Our hope is that these methods will require many fewer training data samples than current approaches.

References

[1] James S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, pages 220–227, 1975.

[2] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 369–376. MIT Press, 1995.

[3] Geoffrey J. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, School of Computer Science, Carnegie Mellon University, June 1999. Also available as technical report CMU-CS-99-143.

[4] Cindy M. Grimm and John F. Hughes. Modeling surfaces of arbitrary topology using manifolds. *Computer Graphics*, 29(2), 1995. Proceedings of SIGGRAPH '95.

[5] Cindy M. Grimm and John F. Hughes. Smooth iso-surface approximation. In *Proceedings of Implicit Surfaces '95*, pages 57–67, 1995.

[6] Paton J. Lewis. Modeling surfaces of arbitrary topology with complex manifolds. Master's thesis, Brown University, 1996.

[7] Andrew W. Moore. Variable resolution reinforcement learning. Technical report CMU-RI-TR, Robotics Institute, Carnegie Mellon University, 1995.

[8] Josep Cotrina Navau and Núria Pla Garcia. Modeling surfaces from meshes of arbitrary topology. *Computer Aided Geometric Design*, 17(2):643–671, 2000.

[9] Josep Cotrina Navau and Núria Pla Garcia. Modeling surfaces from planar irregular meshes. *Computer Aided Geometric Design*, 17(1):1–15, 2000.

[10] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.