Online Value Function Improvement

Mitchell Keith Bloch
University of Michigan
2260 Hayward Street
Ann Arbor, MI. 48109-2121
bazald@umich.edu

John Edwin Laird
University of Michigan
2260 Hayward Street
Ann Arbor, MI. 48109-2121
laird@umich.edu

Abstract

Our goal is to develop broadly competent agents that can dynamically construct an appropriate value function for tasks with large state spaces so that they can effectively and efficiently learn using reinforcement learning. We study the case where an agent's state is determined by a small number of continuous dimensions, so that the problem of determining the relevant features corresponds roughly to that of determining the appropriate level of discretization of the continuous values. We adopt hierarchical tile coding, which applies state aggregation at multiple levels of state abstraction simultaneously. Using our formulation, it is possible to capture the advantages of learning with state abstractions ranging from general to specific using linear function approximation. We then develop a novel algorithm for incrementally refining the degree of state abstraction, based on cumulative absolute temporal difference error, which produces a sparse non-uniform tile coding. We empirically evaluate our approach in the Puddle World and Mountain Car environments. The results demonstrate that the static and incremental hierarchical tile codings significantly outperform individual tilings and multilevel tile codings (CMACs) for initial learning. Our results also indicate that the incrementally constructed tilings perform nearly as well as the full hierarchical tile coding while requiring an order of magnitude fewer weights.

Keywords: Reinforcement Learning, Online Learning, Incremental Learning, Value Function, Tile Coding

Acknowledgements

We acknowledge the funding support of the Air Force Office of Scientific Research, contract FA2386-10-1-4127.

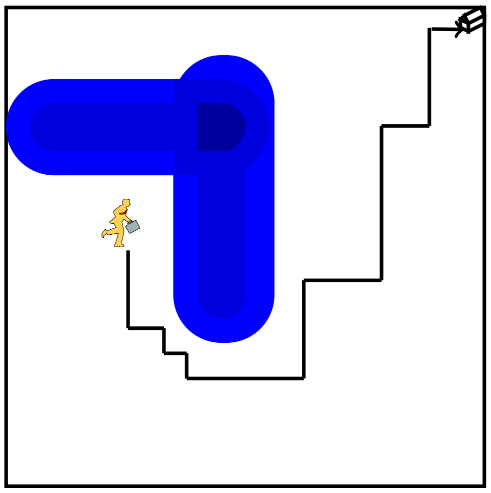
1 Introduction

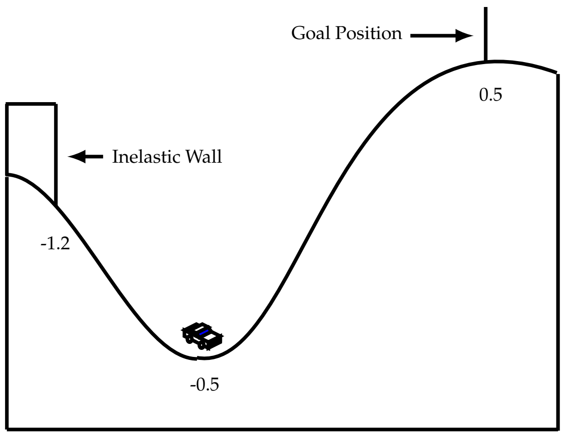
At a broad level, our goal is to build agents which can perform difficult tasks in environments with large state-spaces that are described by a large number of features, some of which may be continuous. In this work, we focus on continuous features and explore a novel strategy for determining when to refine a tile coding[1] in order to allow an agent to improve its policy. We then demonstrate that a tile coding consisting of multiple fixed tilings of variable resolution can do significantly better than any single tiling. We develop incremental hierarchical tile codings that can do nearly as well as static hierarchical tile codings, while using significantly less memory.

2 Environments

We experiment with Puddle World and Mountain Car—two environments with infinitely large state-spaces due to their continuous features. These problems present the difficulty that different parts of their state-spaces warrant reasoning at different levels of precision.

Figure 1(a) shows an example of Puddle World, where an agent can move North, South, East, or West, and where the goal is for an agent to move from an initial location to a goal region. [Sutton, 1996] The world contains "puddles" that are capsule shaped regions whose depth increases from their edges to their centers. The environment is fully observable, but the agent's steps are stochastic, resulting in step sizes between 0.04 and 0.06 units. As the x and y positions are real-valued, the state-space is infinitely divisible, and therefore not perfectly discretizable. The agent receives a penalty of -1 for each step and an additional penalty proportional to the depth of each puddle at its current location.


(a) Puddle World.



(b) Mountain Car.

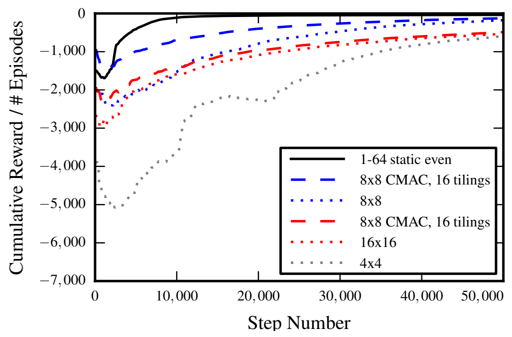
Figure 1: Depictions of our environments.

Figure 1(b) shows the canonical Mountain Car [Singh *et al.*, 1996], where an agent can control the car's motor (left, idle, right), and where the goal is for the agent to move the car from the basin, at rest, to the top of the mountain on the right. Given gravity $-0.0025\cos(3x)$, and a car with power 0.001, the car is incapable of climbing the mountain starting from rest. It is essential to build up potential energy by backing up the hill on the left before moving to the right. The agent receives a penalty of -1 for each step.

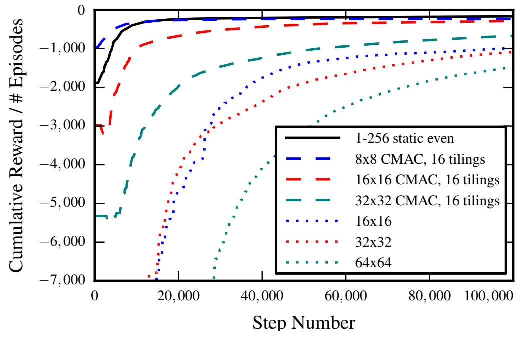
3 Static Hierarchical Tile Coding

We examine hierarchical tile codings where there can be multiple non-overlapping tilings at different resolutions, such as 2x2, 4x4, 8x8, ..., the goal being to support learning at different levels of abstraction. In strict hierarchical tile coding, there are n levels of tilings, with separate tile codings for each action. This introduces a credit assignment problem for hierarchical tile codings, which we solve with an even credit assignment strategy, as is typical for linear function

[1]A tile coding or CMAC (Cerebellar Model Articulation Controller) consists of one or more tilings that partitions a continuous space into a fixed number of a regions, each corresponding to a binary feature. For a detailed explanation, see section 8.3.2 of Sutton and Barto [1998].



(a) Puddle World.



(b) Mountain Car.

Figure 2: Performance for several agents using single tilings, traditional CMACs, and a static hierarchical tile coding.

approximation. General tilings (such as 2x2) receive updates frequently, allowing them to converge quickly. They act as a baseline to speed learning in the more specific tilings (such as 64x64).

We explored hierarchical tile codings with varying subsets of the tilings, 1x1-64x64, such as omitting the most specific or the most general tilings, and none achieved better performance than using the complete hierarchy. Zheng *et al.* [2006], Grzes and Kudenko [2008], and Grzes [2010] previously explored using only two tilings with varying state abstraction.

In our experiments we use an epsilon-greedy exploration strategy ($\varepsilon = 0.1$ for Puddle World and $\varepsilon = 0.01$ for Mountain Car) with a random tiebreak, Q-learning with a learning rate of 0.1 in Puddle World and 1.0 in Mountain Car, and a discount rate of 0.999, and we initialize weights to 0. Figure 2(a) shows the results of using specific tilings for Puddle World, together with traditional CMACs (consisting of multiple identical tilings with different offsets), and hierarchical tile codings. We compare against not only the best CMACs, but also the CMACs corresponding to our single tilings, in order to demonstrate the performance degradation that still occurs as the tile sizes decrease. The y-axis shows cumulative reward per episode, with the x-axis showing total steps. Each data point is an average of 20 runs. We ran experiments with agents using 32x32 and 64x64 tilings; however, they did not start to converge until $> 50,000$ steps and so are not included in the figure. Figure 2(b) shows corresponding results for Mountain Car.

The most dramatic feature of the figure is that the hierarchical tile coding does significantly better than any individual tiling. One hypothesis that these results dispel is that the advantage of the hierarchy is just in hedging the bet as to which tiling is best. Instead, it does much better than even the best single tiling (8x8). The hypothesis it supports is that it can take advantage of the fast learning possible with the more general tilings because of their more frequent updates, while taking advantage of the accuracy provided by the more specific tilings. Moreover, the more specific tilings (such as 32x32 and 64x64) do not drag down the rate of learning.

Additionally, hierarchical tile coding performs better than CMACs for the vast majority of CMAC parameter settings we tried. The hierarchical tile codings dominate all of the CMACs significantly in Puddle World, and most CMACs significantly in Mountain Car. The 8x8 CMAC with 16 tilings does nearly as well in Mountain Car, but only achieves performance comparable to a 1-64 hierarchical tile coding, and a parameter sweep was required to discover it.

So why does hierarchical tile coding work so well? One important feature of both of these environments is that there is continuity in the mapping from the feature space to the weights or Q-values—entries in the value function that are near each other spatially tend to have similar values.

4 Dynamically Refining the Value Function

Although hierarchical tile coding is very effective for these domains, there are two significant problems. First, it requires committing to a set of tilings from the beginning of the task. This is not always be possible for arbitrary tasks. The second problem is that it requires large memories to hold weights for every tiling of the hierarchy, growing exponentially with each additional tiling of the hierarchy. In the future, we plan to study domains with more features where it will be impossible to store weights for every tiling. Thus, we want to develop approaches where the agent does not need to commit to a specific level of tiling, and where the number of weights that must be maintained is minimized.

Incremental approaches to expanding the hierarchy have the potential of satisfying both of these criteria. In incremental approaches, the agent starts with only the most general tilings, which in this case include both a 1x1 and a 2x2 discretization of the task features. Based on experience, the agent determines when it might be useful to have a more specific state abstraction (or more refined discretization) of one of the tiles. An additional level of tiling is created that covers just the chosen tile. Those new tiles are initialized to 0, and with experience, they are updated to match the differences at their level of detail. Additional tiles are expanded, leading to a non-uniform tiling of the space.

The approach explored by Geramifard *et al.* [2011] tracks a fringe of feature conjunctions and expands the set of weights over time. It makes decisions about whether to refine the value function using a static criterion based on TD (temporal difference) error in the fringe, rather than a globally relative criterion like that employed by Munos and Moore [1999] and ours. Additionally it is uses much more memory than our approach given the use of a fringe, and because it may track weights for the full power set of features, rather than using an approach based on a decision tree. For that reason, our approach better satisfies our efficiency criteria.

Inspired by *Stdev_Inf* [Munos and Moore, 1999], the metric we choose to encompass these properties is cumulative absolute temporal difference error (CATDE). TD error—the delta essential to temporal difference methods—is highest in regions of high variance. Tracking the CATDE for a tile in parallel with each weight provides a metric which increases more quickly when the variance associated with taking an action is high, and when that action is taken frequently.[2] Environmental stochasticity artificially inflates the CATDE values but, because the variance is estimated locally, the impact on our metric is less than the impact on *Stdev_Inf*. Additionally, CATDE can be tracked incrementally, requiring low computational costs.

Given the CATDE metric, we choose to select the tiles with the greatest CATDE for refinement. In order to make this decision procedure incremental, it is essential to have an efficient algorithm for tracking the mean and variance for CATDE throughout our value functions. Incrementally tracking a mean for a set of values is fairly trivial, but incrementally calculating the variance requires a modified version of an algorithm provided by Knuth and Welford [Knuth, 1997; Welford, 1962]. We implemented additional methods to allow updating values and removing values from the set.

Whenever a tile is refined, the CATDE for its region is reset to 0. Initially, the CATDE metric will result in significantly faster refinement to some regions than others. But as time goes to infinity, tiles will tend to split at the same rate, as regions which receive greater refinement will accumulate TD error over smaller subsets of the state-space over time.

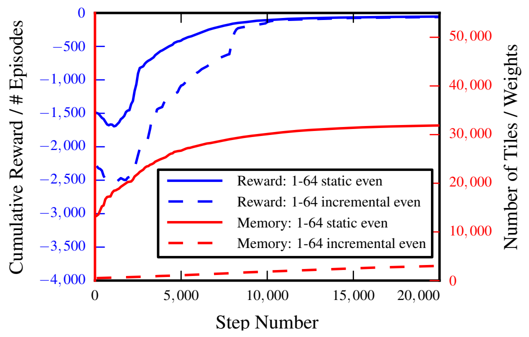
Given a tile with CATDE $= c$, our mean estimate, μ, and our variance estimate, σ^2, we refine a tile if

$$c > \mu + z\sigma^2 | z = 0.5 \tag{1}$$

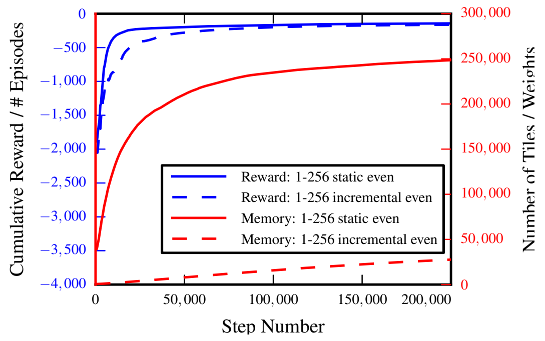
and the tile has not been visited in the past 20 steps. z is chosen to determine how selective the agent should be in choosing which tiles to refine. The 20 step threshold prevents overzealous refinement.

Given the limited number of features in both Puddle World and Mountain Car, our agents simply alternate back and forth between the dimensions when making these refinements.

Figure 3(a) shows data for the Puddle World domain. Data for static hierarchies, with tilings of resolutions 1x1 through 64x64, and for incremental hierarchies, with tilings of resolutions initially between 1x1 through 2x2 are presented together. The number of weights is overlaid over the performance data so that one may compare their performance to their memory efficiency as time passes. The cumulative performance of the incremental hierarchies is within 13% of the static hierarchies by 20,000 steps, however the number of weights is reduced by 90%. Figure 3(b) shows corresponding data for the Mountain Car domain. Data for static hierarchies, with tilings of resolutions 1x1 through 256x256, and for incremental hierarchies, with tilings of resolutions initially between 1x1 through 2x2 are presented together. The cumulative



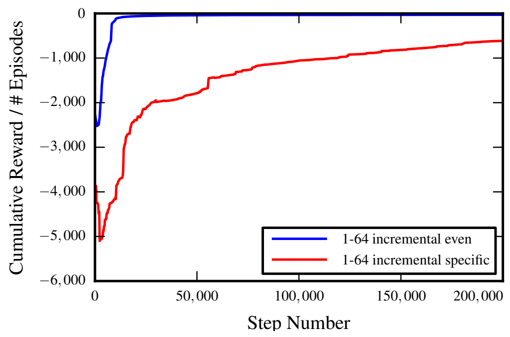
(a) Puddle World.



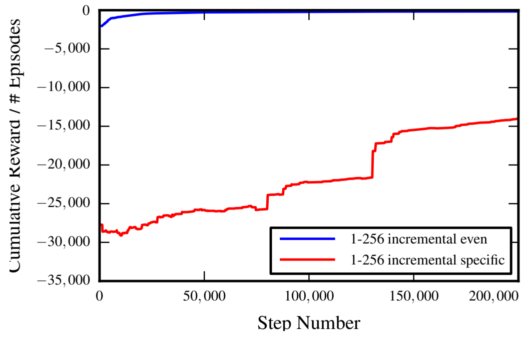
(b) Mountain Car.

Figure 3: Performance and memory usage of an agent using a static hierarchical tile coding and another agent using an incremental hierarchical tile coding.

[2]It is critical that CATDE is cumulative. If it were non-cumulative, it would essentially eliminate the idea of influence from *Stdev_Inf*, and turn it into a kind of variance metric. This has been confirmed empirically (not shown).



(a) Puddle World.



(b) Mountain Car.

Figure 4: Performance of agents using incremental hierarchical tile codings with different credit assignment strategies.

performance of the incremental hierarchies is within 14% of the static hierarchies by 200,000 steps, however the number of weights is reduced by 89%.

Figure 4 contrasts our incremental hierarchical tile coding against a tile coding which gives all credit to the most specific tiles. This is equivalent to comparing against an adaptive tile coding which splits tiles, keeping only one tile per region, as described by Munos and Moore [1999] and Whiteson *et al.* [2007]. The performance of the even credit assignment strategy dominates the performance of the specific credit assignment strategy. The results of these experiments are presented in figure 4 with a significant change in scale for both the x and y-axes.

5 Conclusion

In summary: we demonstrated that static hierarchical tile codings dominate individual tile codings and CMACs in two domains; we developed an incremental hierarchical tile coding which performs well while saving memory; and we demonstrate that incremental hierarchical tile codings dominate incrementally split, non-hierarchical tile codings.

References

[Geramifard *et al.*, 2011] Alborz Geramifard, Finale Doshi, Josh Redding, Nicholas Roy, and Jonathan P. How. Online discovery of feature dependencies. In Lise Getoor and Tobias Scheffer, editors, *ICML*, pages 881–888. Omnipress, 2011.

[Grzes and Kudenko, 2008] Marek Grzes and Daniel Kudenko. Multigrid reinforcement learning with reward shaping. In *ICANN (1)*, pages 357–366, 2008.

[Grzes, 2010] M. Grzes. *Improving exploration in reinforcement learning through domain knowledge and parameter analysis*. PhD thesis, University of York, 2010.

[Knuth, 1997] Donald E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition)*. Addison-Wesley Professional, 3 edition, November 1997.

[Munos and Moore, 1999] Remi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine Learning Journal*, 1999.

[Singh *et al.*, 1996] Satinder Singh, Richard S. Sutton, and P. Kaelbling. Reinforcement learning with replacing eligibility traces. In *Machine Learning*, pages 123–158, 1996.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. Reinforcement learning i: Introduction, 1998.

[Sutton, 1996] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996.

[Welford, 1962] B. P. Welford. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics*, 4(3):419–420, 1962.

[Whiteson *et al.*, 2007] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Adaptive tile coding for value function approximation, 2007.

[Zheng *et al.*, 2006] Yu Zheng, Siwei Luo, and Ziang Lv. Control double inverted pendulum by reinforcement learning with double cmac network. *Pattern Recognition, International Conference on*, 4:639–642, 2006.