Online Value Function Improvement

Mitchell Keith Bloch

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

June 3-7, 2013
Primary objective is to learn how to act, or to derive an optimal policy.

Prefer actions leading to positive rewards to actions leading to negative rewards.

Outcomes are characterized as a discounted return, \( \sum_{t=0}^{\infty} \gamma^t r_t \).

Deriving good estimates of these returns for different actions is essential for many RL algorithms.

See [Sutton and Barto, 1998] for an excellent primer.
Temporal Difference Method: Q-Learning

Given
- a discount rate, $\gamma$
- a Q-function, $Q(s, a)$, to represent value estimates for state-action pairs, and
- an immediate reward, $r$,

the update rule is expressed:

$$Q(s, a) \leftarrow r + \gamma \max_{a^*} Q(s', a^*)$$

Without approximation, all $Q(s, a)$ values are independent.
- This uses $O(|s| \times |a|)$ memory.
- This doesn’t support generalization.
A tile coding partitions the state-space, providing a coarser representation.

The CMAC (Cerebellar Model Articulation Controller) is the traditional approach to using multiple tile codings. [Sutton, 1996]
Soar-RL provides Q-learning and Sarsa [Nason and Laird, 2004]

Conditions on RL-rules encode which features to test and how to discretize continuous state, defining the mapping $S \times A \Rightarrow Q$
  - Can be one-to-one (if there no continuous features)
  - Can use coarse coding, effectively implementing tile coding
  - Potentially arbitrary, non-uniform abstraction

Typical generalizations in Soar-RL rules effectively implement one or more tile codings
Motivation

We’re concerned with the problem of generating a value function capable of supporting the computation of a near-optimal policy for a task with

- a large state-space
- composed of many features,
- some of which may be continuous.

We’re additionally concerned with problems of

- efficient learning,
- computational limitations,
- and memory limitations.
We have broken down the problem into a number of subproblems:

1. Large, Sparse State-Spaces
2. Combining Values from Hierarchical/Overlapping Tilings
3. Credit Assignment for Hierarchical/Overlapping Tilings
4. Deciding When and Where to Refine the Value Function
5. Deciding How to Refine the Value Function
6. Complexities of These Approaches
Problem 1: Large, Sparse State-Spaces

Many agents developed using cognitive architectures operate in environments with

- large state-spaces,
- state-spaces described by large numbers of features, or
- continuous features which cannot be perfectly discretized.

Thankfully,

- the portion of the environment an agent must explore is often a relatively small subset of the state-space,
- features are not totally independent from one another,
- and satisfactory discretizations can usually be found.

Our strategy: hierarchical tile coding
Problems We're Looking At

Puddle World

Goal: Get to the upper-right corner, avoiding the puddles if possible.

2-dimensional state-space

Continuous-valued features

Four actions: North, South, East, and West

Stochastic movement

See [Sutton, 1996].
What Does a Hierarchical Tile Coding Look Like?

A partial tiling for the “move North” action in Puddle World:
Problems We’re Looking At

Problems 2 & 3: Hierarchical/Overlapping Tilings

Combining Values:
- Summation is typical (i.e. linear function approximation).
- This works for statically and dynamically generated tilings.

Credit Assignment:
- The standard approach has been even credit assignment between tiles.
- We consider alternatives which shift credit from more general tilings to more specific tilings over time.
Linear Function Approximation

Using

- $n$ weights, and
- a Boolean function, $\phi_i(s, a)$, to determine whether to include any given weight

$Q(s, a)$ can be calculated:

$$Q(s, a) = \sum_{i=1}^{n} \phi_i(s, a)w_i,$$

(2)

This can reduce memory usage substantially.

Done well, this may also support efficient generalization from experience.
Performance for several agents using single tilings, and one using a static hierarchical tiling, in Puddle World:
Mountain Car

Goal: Get to the top of the hill.

2-dimensional state-space

Continuous-valued features

Three actions: Accelerate left, idle, and accelerate right

Some dynamics

See [Moore, 1991].
Performance for several agents using single tilings, and one using a static hierarchical tiling, in Mountain Car:
Problems 4 & 5: Refining the Value Function

When and Where:

- Must determine when and where the value function is not sufficiently specific to represent a near-optimal policy
- Must do this online, in an incremental fashion
- Must cope with error due to environmental stochasticity

Our criterion: **Cumulative Absolute Bellman Error**

How:

- Must determine which features would be most beneficial to consider
- Must increase refinement of discretizations
- Must do this online, in an incremental fashion, without using a great deal of memory storing a model or instances
Static vs Dynamic (Hierarchical): Puddle World

Results for one agent using a static hierarchical tiling and another agent using an incremental hierarchical tiling in Puddle World:

Performance:

The number of weights:
Results for one agent using a static hierarchical tiling and two agents using incremental hierarchical tilings (one with even credit assignment, and one with $1/\ln(\text{update count})$ credit assignment) in Mountain Car:

Performance:

The number of weights:
Problem 6: Complexities

Environmental:
- Environmental stochasticity
- Propagation delays / Mixing time
- Partial observability
- State aliasing

Keeping the value function small for
- savings in computation time and
- memory usage.
Other Environments

We wish to work more with additional environments:

- **Equilibrium Tasks**: 2 and 4-dimensional versions of Cart Pole
- **Relational Domains**: Blocks World
- **Future Work**: The above, and additionally Liar’s Dice

We plan to

- improve our refinement criterion,
- add support for automatic feature selection, and
- focus more on the tradeoffs between computational and memory costs and learning efficiency.
Nuggets:
- We have an efficient codebase to experiment with.
- We have demonstrated the efficacy of deep hierarchical tile codings.
- We have shown that alternative credit assignment strategies have promise.
- Work so far is consistent with the implementation of Soar-RL.

Coal:
- Our current refinement/splitting criterion doesn’t work very well in certain domains.
- The most recent experiments are not being done in Soar.

