Adaptive Value Estimate Updates in Generalized Policy Iteration Dynamic Programming for Reinforcement Learning

John Ahmed Dellas[1], Joseph Anderson, PhD[2]
[1]Rutgers University-NB, [2]Salisbury University

Abstract

Most modern "adaptive" methods of accelerating traditional dynamic programming policy iteration algorithms rely on using a neural net to approximate the value function [1]. In this proposal, we instead focus on accelerating DP by adaptively selecting the order in which state value estimates are updated using other reinforcement learning algorithms so as to minimize convergence time.
More specifically, we tackle the classical DP-RL problem of finding the best policy iteration strategy by exploring the use of a Monte Carlo Reinforcement Learner to accelerate the convergence of Dynamic Programming Policy Iteration algorithm. The MC Learner is tasked with finding the optimal "sweep" through the state space during policy evaluation. We analyze the shortcomings of using an MC method for this purpose as originally proposed and present alternatives to extending the effectiveness of a crude proof-of-concept algorithm implemented and tested on a classical DP-RL problem.

Introduction

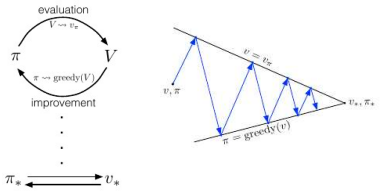
Reinforcement Learning refers to a set of algorithms, techniques and methods used to train some agent to accomplish a goal-driven task by having the agent maximize a reward that it receives by interacting with its environment. This interaction is typically represented by 3 signals which pass between agent and environment: state, action and reward. The *state* signal can be thought of as an indication of where the agent is in the set of all possible arrangements of its environment (a state in a game of chess might refer to an arrangement of the pieces on the board) that can be reached by performing some *actions* (the second signal, passed from agent to environment). The agent accomplishes its task by following a *policy*. A policy, denoted using the lowercase pi, is a mapping from the set of all actions (referred to as the action space) to probabilities that the agent selects that action given that it is currently at a particular state. More specifically, the goal of most RL algorithms to obtain the *optimal policy* (optimality in general is denoted with a subscript "*"), which is the only policy that is deterministically greedy with respect to its value function. There are two flavors of value functions: state value functions (denoted with a "v"), and action value functions (denoted with a "q"). A state value function of a policy gives the expected return by starting at a particular state and following the policy associated with it thereafter (action value functions are identical except that instead we start at a state-action pair). Return is defined as cumulative function of the rewards experienced after a time step t, and is typically denoted G_t. The value functions of the optimal policy are called the optimal value functions, and are the unique functions which satisfy the *Bellman Optimality equations,* which are just the *Bellman equations* for the optimal policy (and can therefor be written in a special form without reference to any specific policy). The Bellman equation for the state value function (1) gives the value of one state in terms of the values of its successor states. *Dynamic Programming,* as applied to RL, refers to a collection of algorithms used for solving RL problems when the dynamics of the environment are completely known. The most popular DP-RL algorithm, *policy iteration*, is the focus of this research. Policy Iteration (outlined in (2)) works by turning the Bellman equation into an update rule and exploiting the fact that iterating value function estimates produced by the rule will converge to the true value function of a given policy at infinity. Standard policy iteration has two main components: *policy evaluation* and *policy improvement*. Policy evaluation is the calculation of the value function of a policy given as input. Policy improvement is the creation of a new policy that is greedy with respect to the newly computed value function to produce a better policy which can then be fed back into policy evaluation until the optimal policy is produced. Policy comparison is defined as:

$$\pi_1 \leq \pi_2 \leftrightarrows v_{\pi_1}(s) \leq q_{\pi_1}(s, \pi_2(s)) \forall s$$

Note that the order in which state values are updated during policy iteration significantly influences the rate of convergence of the algorithm [2]. Generalized policy iteration considers versions of policy iteration in which state value updates are asynchronous and intermixed with policy improvements.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s'] \Big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}, \end{aligned}$$

1. Bellman equation for the state value function.



2. Right: A set of illustrations of Generalized policy iteration In action

Approach

- Traditional implementation of DP-PI has arbitrary state space sweep.
- However, sweep selection can *significantly* influence convergence.
- **The task of selecting sweeps through the state space during DP policy evaluation can be modeled as a FMDP!**
  - States: States of the underlying MDP
  - Actions: The decision of which state to evaluate next
  - Rewards: The sum of the differences between the previous state value estimate and the new state value estimate across estimates which have yet to fall beneath the threshold
  - Discounting Rate: gamma = 1 (episodic)
- An episode in the task of finding the best sweep would be n policy evaluations before episode termination.

Using a "supervising" reinforcement learning algorithm, we can intelligently select which states we choose to update next, and reward update sweeps that maximize the rate of convergence of the underlying DP algorithm. Originally, we intended to implement this supervisor with a Monte Carlo Learning method. However, it became clear with time that a MC method would be inappropriate, and was dropped in favor of a Temporal Difference learning method based approach.

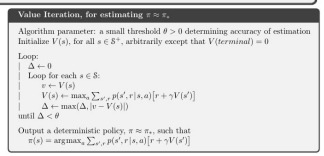
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

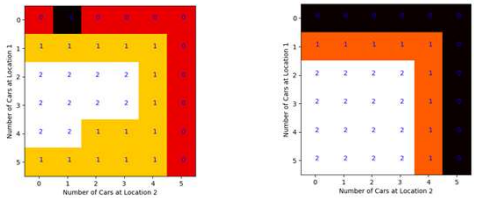
3. Top: Standard Policy Iteration
Right: Value Iteration, an example of GPI. Used in Jack's Car Rental Prob.



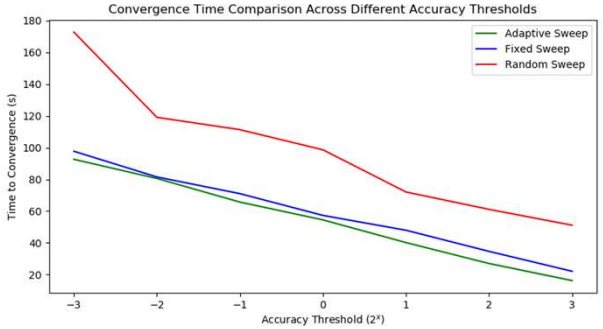
Performance Evaluation

We rely on a classical DP problem known as Jack's Car Rental problem to evaluate crude adaptive sweep performance against traditional arbitrary sweeps.

- Max Number of Cars at each location: 5
- 10$ for rental, -1$ for transportation
- Max Number of cars transported overnight: 2
- Discounting Rate: 0.9
- Car transactions per day modeled by Poisson random variables (# of cars returned at location 1: mean 3; # of cars rented at location 1 and returned at location 2: mean 1; # of cars rented at location 2: mean 4)



4. Left: Random policy after 1 iteration. **Right:** Optimal policy after convergence.



Improvements and Conclusions

- Memory Complexity is not sustainable for problems of any appreciable size
  - Look into non-tabularized, approximation based methods for value function representation, such as neural nets.
- DP sweeps are not an inherently episodic task
  - MC methods are most appropriate for episodic tasks
  - Needing to arbitrarily stop DP execution to perform the backwards computation required for MC action value updates is both silly and slows the convergence of the algorithm
  - Using an on-line learning algorithm such as TD Learning allows for experiential learning *with* bootstrapping (no costly backwards return computations like in MC methods).
- Crude implementation for Jack's Car Rental problem does not allow for experimentation with multiple policy evaluation steps before a policy improvement
- Full implementation would require more complex reward signal
- Eventually, we will tackle more interesting RL problems
  - Specifically problems with asymmetric action spaces

Contact

John Ahmed Dellas
Rutgers University-New Brunswick
jad525@scarletmail.Rutgers.edu
609-608-2057

References

1. F. Wang, H. Zhang and D. Liu, "Adaptive Dynamic Programming: An Introduction," in *IEEE Computational Intelligence Magazine*, vol. 4, no. 2, pp. 39-47, May 2009.
2. Sutton and Barto, "Reinforcement Learning: An Introduction", 2nd Ed