Overview of Statistical Machine Learning

Doug Aberdeen

November 4, 2004

Updates

- Now consistent with use of δ as discount (in slides) instead of old β.

Assignment 4: $SARSA(\lambda)$

This assignment is a straight forward implementation of the tabular $SARSA(\lambda)$ algorithm. The domain is a simple 'grid world': an $X \times Y$ Cartesian grid illustrated by Figure 1. A robot at state (x, y) can move around the grid using four actions: north (y increasing), south (y decreasing), east (x increasing), or west (x decreasing). The robot's task is to move along the shortest path from a random initial position to the goal position (x_g, y_g).

We will model some noise in the robot's movement. Whatever direction the action specifies, there is a:

- 0.85 probability of moving 1 unit in that direction;
- 0.1 probability of not moving;
- 0.05 probability of moving 2 units in that direction.

If the robot bumps into a wall it should stay in the position where it bumped into the wall. For example, if the robot is in state $(x, 1)$, and it chooses to move south, then it has probability $0.85 + 0.05 = 0.9$ of ending in state $(x, 0)$ and probability 0.1 of staying in state $(x, 1)$. These probabilities are not used by the $SARSA(\lambda)$ algorithm directly, but they are necessary for you to construct the simulator which $SARSA(\lambda)$ needs to generate sample state/action trajectories.

The reward is 0 for all states except (x_g, y_g) where the reward is 1. Reaching the goal terminates the episode. The new episode begins with the agent

Algorithm 1 SARSA(λ).

Given:

- SARSA policy $\Pr[a|s, \mathbf{w}]$
- $\delta, \lambda \in [0, 1)$ (λ may be 1 for episodic tasks)
- Step size α
- Arbitrary starting state s_0, and initial action a_0 sampled from $\Pr[a|s, \mathbf{w}]$
- $e(s, a) = 0$

for each transition $(s, a) \rightarrow s', a'$ generated according to the MDP and policy $\Pr[a|s, \mathbf{w}]$ **do**
  Compute $d(s, s') = r(s) + \delta Q(s', a') - Q(s, a)$
  $e(s, a) \leftarrow e(s, a) + 1$
  for all $(\bar{s}, \bar{a})$ **do**
    $Q(\bar{s}, \bar{a}) \leftarrow Q(\bar{s}, \bar{a}) + \alpha d(s, s') e(\bar{s}, \bar{a})$
    $e(\bar{s}, \bar{a}) \leftarrow \delta \lambda e(\bar{s}, \bar{a})$
  end for
  $s \leftarrow s'$; $a \leftarrow a'$
end for

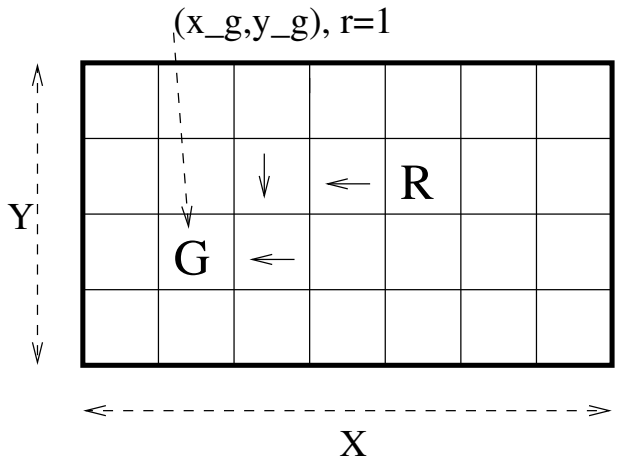

Figure 1: A 7×4 grid world, with the robot located in state (5,3) and the goal located in state (1,1). One possible shortest route to the goal is illustrated.

located in a random state (chosen uniformly) excluding the goal state. Don't forget to reset the eligibility trace at the end of each episode.

Different levels of exploration and different discount factors will result in different values, but the greedy policy should be an optimal policy given sufficient episodes. Don't get too fancy with policies that reduce exploration over time. It can be hard to get right and should not impact the optimality of the final *greedy* policy.

You can use any reasonably common language, including Matlab. Your program (or Matlab function) should be called `sarsa` and expect the following parameters in order:

- integers X, Y: size of the grid world;
- integers x_g, y_g: location of the goal state;
- floating point $\lambda \in [0, 1]$: $SARSA(\lambda)$ parameter;
- floating point $\delta \in [0, 1)$: discount factor;
- floating point $\alpha \in [0, \infty)$: step size;
- integer E : number of episodes to run.

For example,

```
sarsa 20 20 1 1 0.5 0.95 0.01 1000
```

means run 1000 episodes of $SARSA(0.5)$ in a 20×20 grid world, with the goal located at state $(1, 1)$ and a discount factor of $\delta = 0.95$, step size $\alpha = 0.01$. Your program should produce some output at the end of each episode as a progress indicator. After the episode limit is reached your program should output a line for each state

```
V(x,y) = v a
```

where x, y are the state coordinates, v is the value, and a is the greedy action for that state (which is not necessarily the policy you implemented for $SARSA(\lambda)$ to follow). For example

```
V(0,0) = 0.9025 north
V(0,1) = 0.95 east
⋮
```

You should submit source code,[1] a Makefile (if necessary), and a text file

[1] I'll be running code on a Linux machine.

called readme.txt containing any extra instructions for running your code and brief answers to the following questions:

- Which exploration method did you employ?
- The task is episodic so we can compute long-term averages instead of discounted sums, but what happens if we use discount $\delta = 1$ in this domain?
- Using this reward function, and given a particular discount factor, what meaning can we attach to the value of each state?

See `http://www-anw.cs.umass.edu/~rich/book/the-book.html` for information about $SARSA(\lambda)$. Please contact me (Doug) for help or clarification. If you are confused, odds are everyone else is as well.