AI - Gridworld - Dynamic Programming Demo

stefan.mojsilovic

November 12, 2017

1 Deterministic Problem Setup

Gridworld is a toy environment for simple Reinforcement Learning (RL) methods. This demo is realized using the "REINFORCEjs" library https://github.com/karpathy/reinforcejs under the MIT licence.

1.1 World

The problem is a finite Markov Decision Process (MDP) which is defined by the following four parts:

State space - $s \in S$: Gridworld is a "room" represented as a 10x10 grid of cells. Cells can be either regular cells that are part of the state space or walls that are not part of the state space. Borders of the gridworld (outside the 10x10 grid) are considered to be walls. There are two special states, the start state and the goal state.

Actions - $a \in A(s)$: A reinforcement learning agent has four possible actions to chose from: Up, Down, Left, Right, which indicate the desired movement from the current state s towards the next state s' in the grid. Some of those actions are unavailable if the next cell in the respective direction is a wall. Agent starts from the start state by chosing one of the available actions. Agent teleports to the start state from the goal state no matter which action it chooses while in the goal state.

Transitions - $P(s, a, s')$: Probabilities that the process moves into a next state given the previous state and the chosen action. In this demo actions are deterministic, they always result in only one next state which corresponds to the direction of the chosen action.

Rewards - $R(s, a, s')$: Rewards for the agent are real numbers. In our case they depend only on the state the agent arrives to. Reward for the goal state is predefined to be 1.

1.2 User Interface

A simple problem is already defined. Start state is in the top left corner cell [0,0]. Goal state is in cell [5,5]. There are two walls in cells [4,5] and [5,4]. There is a cell with negative -1 reward in cell [6,5].

The User can select any cellby clicking on it and change it from a regular cell to a wall cell or vice versa by the "Wall/Regular" button. A regular cell can be changed to a start cell by the "Set as Start" button or a goal cell by the "Set as Goal" button. Rewards for regular cells can also be changed by using the slider. Reward for the goal cell is predefined to be 1.

After setting up the grid, the User can start the dynamic programming solver one iteration at a time or by toggling on the automatic process see the following section for details. Reset button resets the whole grid to the starting state.

1.3 Dynamic programming

(TODO To be written out in more detail according to the Chapter 4 from Sutton's Book:
http://ufal.mff.cuni.cz/ straka/courses/npfl114/2016/sutton-bookdraft2016sep.pdf)

Bellman Equation:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right]$$

(TODO: Taken from the reinforcejs website, to be edited in future)
With our goal of finding the optimal policy $\pi^*(s,a)$ that gets the most Value from all states, our strategy will be to follow the Policy Iteration scheme: We start out with some diffuse initial policy and evaluate the Value function of every state under that policy by turning the Bellman equation into an update. The value function effectively diffuses the rewards backwards through the environment dynamics and the agent's expected actions, as given by its current policy. Once we estimate the values of all states we will update the policy to be greedy with respect the Value function. We then re-estimate the Value of each state, and continue iterating until we converge to the optimal policy (the process can be shown to converge). These two steps can be controlled by the buttons in the interface:

- The Policy Evaluation (one sweep) button turns the Bellman equation into a synchronous update and performs a single step of Value Function estimation. Intuitively, this update is diffusing the raw Rewards at each state to other nearby states through 1. the the dynamics of the environment and 2. the current policy.
- The Policy Update button iterates over all states and updates the policy at each state to take the action that leads to the state with the best Value (integrating over the next state distribution of the environment for each action).
- The Value Iteration button starts a timer that presses the two buttons in turns. In particular, note that Value Iteration doesn't wait for the Value function to be fully estimates, but only a single synchronous sweep of Bellman update is carried out. In full policy iteration there would be many sweeps (until convergence) of backups before the policy is updated.