TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Datenverarbeitung
Prof. Dr.-Ing. K. Diepold

Approximate Dynamic Programming & Reinforcement Learning

Programming Assignment

December 5, 2017

Introduction

In this programming assignment you will implement a simple environment consisting of a maze. The goal in this environment is to reach a goal state from some starting state. We now want to find the optimal policy maneuvering the agent through the maze. Optimal in this case means taking the least steps.

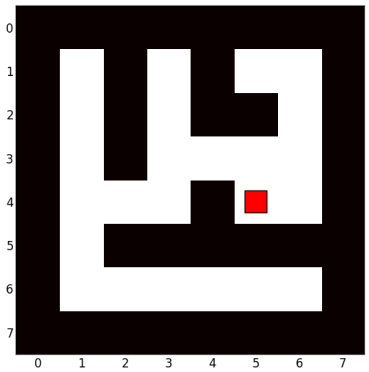
You have to use Python (version 2.7 if possible) for the implementation. You are allowed to use basic modules like Numpy for representing vectors or Matplotlib for plotting, but no existing RL/DP framework (e.g. the Reinforcement Learning Toolkit by Richard Sutton).

The Environment

As the maze is a very simple environment, we can represent the dimensions in a matrix, where a 0 indicates a cell, where the agent can go and a 1 represents a wall.

```
# maze (0 = free, 1 = wall)
maze = [[1, 1, 1, 1, 1, 1, 1, 1],
        [1, 0, 1, 0, 1, 0, 0, 1],
        [1, 0, 1, 0, 1, 1, 0, 1],
        [1, 0, 1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 1, 0, 0, 1],
        [1, 0, 1, 1, 1, 1, 1, 1],
        [1, 0, 0, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 1, 1, 1]]

# start and goal position in the maze
start = [4, 5]
goal = [6, 6]
```



The agent has four actions to move within the environment $U := \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}$. Once the agent transitions into the goal state, the episode is over and the agent cannot move any further. Any transitions from the goal state to itself are at no cost. Your tasks:

- Adapt the maze for your programming environment.
- Implement the functionality to move the agent around and record its position. If the agent manages to move into a field occupied by a wall it is sent back to the field where it came from.

Probabilistic Formulation of the Problem

In order to solve arbitrary Markov Decision Processes with your implementation of the Dynamic Programming algorithms it is useful to allow for a probabilistic formulation of the problem. Thus you have to:

- implement a state transition probability function $p_{ij}(u)$, which can be used to sample all possible successor states j given the current state i and control u
- describe why you decided to implement it in the way you did. What are the problems of a naive implementation of $p_{ij}(u)$ for all possible combinations of i,j and u?

In this maze the ground is a bit slippery, meaning that the outcome of an action varies. Because an oracle has told you the model of the environment, you know that applying an action will result with 70% in the desired next state (e.g. executing *up* will result in the state above the current one in 7 out of 10 cases, assuming that there is no wall) and with 30% nothing happens, meaning the current state and the successor state are identical.

One-step-cost

We have to model the goal-finding with the help of the one-step-costs $g(\cdot)$. In this case there could be several possibilities to communicate the desired behavior via the costs, e.g.:

1. walking through the maze is at no cost, transitioning into the terminal goal state gets rewarded with a treasure, i.e. a negative cost.
2. punishing the wast of energy, i.e. each move in the maze produces a fixed cost of 1.0. Transitioning into the goal state then is at no cost (or at a negative cost, if you like to adapt the idea of a treasure from above).

Implement two cost functions (i.e. callable python functions) $g_1(i, u, j)$ and $g_2(i, u, j)$, which reflect the behavior described above.

Policy

In order to learn a optimal path from a start state to the goal, you need a policy to map each state to an action. You have to:

- think of a way to represent a modifiable deterministic policy $\mu(i)\ \forall i \in \mathcal{S}$
- implement it

Solution with Dynamic Programming

Now you have all the ingredients ready for solving the problem with Dynamic Programming. The last thing, that is missing, is the cost function J itself. To achieve this, think of a way to linearize the two dimensional state space to represent the cost function J as a simple 1-D vector. You tasks:

- Find a mapping to linearize the state space into a vector. What is your solution?
- Implement Value Iteration
- Implement Policy Iteration, i.e.:
  - Policy Evaluation
  - Policy Improvement
- How can you determine if the DP algorithms have converged? Think of a sensible criteria and describe it.
- Implement a visualization for a policy and cost function in a 2d plot. Colored boxes and a heat map should show the entries of J for each cell in the maze and a quiver plot should render the actions produced by the policy.
- Derive the optimal policy with both algorithms for $\alpha = 0.9$ and for both costs g_1 and g_2.
- Do the two methods generate the same policy?
- Vary the discounting factor α (e.g. 0.5 and 0.01) an repeat the steps before. Does this have an influence on the final policy?

A Study of Algorithm Properties

To make this assignment complete, you will study the behavior of the algorithms:

- Run Policy Iteration and Value Iteration until each have converged. Regard the solution of those runs as the ground truth value function and policy.
- Now run the algorithms again and plot the error (i.e. the squared distance) to the ground truth with respect to the iterations.
- Create these error plots for three suitable values of α and for both one-step-cost functions. Suitable α are those where its impact becomes visible and the comparison of PI and VI is easy. Which values for *alpha* did you use?
- Is VI or PI better / faster?

Final Instructions

Summarize your results in a short report and save it as a PDF file. The report has to contain the answers to the all the sentences ending in a question mark on maximal two pages. Then attach the plots. Try to put as many on one page without destroying the visual quality (a 3x2 grid should be fine).

Zip all your source code files and the report to a single `.zip` archive. As filename use *lastname-firstname-matriculationnumber.zip*. and hand this in on Moodle.

You have to provide one file called *main.py*, that can be run to produce the exact plots that you have in your report (don't forget their titles).