605.449 — Introduction to Machine Learning

Programming Project #7

Due: December 9, 2018

For the seventh and final programming assignment, you will implement a reinforcement learner and apply it to the racetrack problem. The racetrack problem is a standard control problem. The goal is to control the movement of a race car along a pre-defined racetrack. You want your race car to get from the starting line to the finish line in a minimum amount of time. In this version of the problem, your agent will be the only racer on the track, so it is more like a time trial than a full competitive race.

More formally, at each time step, the state of the agent can be encoded in four variables: x_t and y_t are the x and y coordinates corresponding to the location of the car at time step t (we treat the racetrack as being layed out on a Cartesian grid). The variables $\dot{x}_t$ and $\dot{y}_t$ represent the x and y components of the car's velocity at time t. The control variables for the car are a_x and a_y, which represent the x and y components of an acceleration vector to be applied at the current time step. The system is governed by an approximation of the standard laws of kinematics:

$$\begin{aligned}
x_t &\equiv x \text{ position}\\
y_t &\equiv y \text{ position}\\
\dot{x}_t &= x_t - x_{t-1} \equiv x \text{ speed}\\
\dot{y}_t &= y_t - y_{t-1} \equiv y \text{ speed}\\
a_{x_t} &= \ddot{x}_t = \dot{x}_t - \dot{x}_{t-1} \equiv x \text{ acceleration}\\
a_{y_t} &= \ddot{y}_y = \dot{y}_t - \dot{y}_{t-1} \equiv y \text{ acceleration}
\end{aligned}$$

At any given time step, your car only has active control over the values of a_x and a_y and must use these control variables to influence the car's state. This essentially gives you the ability to accelerate, decelerate, and turn. There is a further restriction that the set of values that may be assigned to a control variable is -1, 0, and 1. That is,

$$a_{\{x,y\}} \in \{-1, 0, 1\}.$$

The velocity of your car at any given time is limited to $(\dot{x}_t, \dot{y}_t) \in [\pm 5, \pm 5]$. Any attempt to accelerate or decelerate beyond these limits will be ignored.

As an example, if at time $t = 0$ your car is at location $(2, 2)$ with velocity $(1, 0)$, it is essentially moving towards the east. If you apply an acceleration of $(1, 1)$, then at timestep $t = 1$ your position will be $(4, 3)$ and your velocity will be $(2, 1)$. At each time step, your acceleration is applied to your velocity before your position is updated. Strictly speaking, this is not a good model of Newtonian physics, but we're dealing with integer position and velocity values, so some simplifying abstractions need to be made.

If this were the extent of the problem, it would be quite easy to solve for any given track, so to make things a bit more interesting, we are going add in a small amount of non-determinism. To do this, we assign a probability to each attempted action for each possible outcome. For this assignment, we specify that for any attempt to accelerate, there is a 20% chance that attempt will simply fail, and the velocity will remain unchanged at the next timestep. Thus, at each timestep, the probability of accelerating as specified is 80% and the probability of having no acceleration is 20%.

In this problem, there is an additional requirement that you stay on the track; crashing into the wall is bad. You will experiment with two different versions of how "bad" it is to crash. The first variant says that, if the car crashes into a wall, it is placed at the nearest position on the track to the place where it crashed, and its velocity is set to $(0, 0)$. The second, harsher variant says that when a car crashes, its position is set back to the original starting position, as well as zeroing its velocity. Essentially, in this latter case if you crash, you have to start over from the beginning. You should implement both variants in your program so that you can experiment with what effects they have on the strategies your car learns. Since you have a limited time to run experiments, however, you are only expected to do a side by side comparison of the two definitions of a "crash" on the R shaped track (in the file `R-track.txt`). For the other tracks, use the version where crashing stops the car but leaves it in the location it crashed.

The cost function is 1 for each move, except when you reach the finish line. The finish line locations are absorbing states with cost 0. Since you are attempting to minimize cost, this translates to attempting to complete the race in as few time steps as possible. You do not need to *stop* on a finish state. It is sufficient to cross the finish line, so this means the "wall" behind the finish line really isn't a wall.

Here are the specific steps that need to be followed.

- Implement the racetrack problem as described above, including a racing simulator that takes a track specification (described below).
- Implement the Value Iteration algorithm and *either* the Q-learning or SARSA algorithm, and apply them to the racetrack problem.
- For extra credit (for up to 10 additional points), implement *both* Q-learning and SARSA and apply them to the racetrack problem.
- For extra credit (for up to 20 additional points), implement TD-learning with function approximation (i.e., with a feedforward neural network) and apply it to the racetrack problem.
- Test your algorithms on the two of the three provided tracks in the data files described below. One of the tracks must be the R-track, but you can choose between the L-track and the O-track. The files are named `L-track.txt`, `O-track.txt`, and `R-track.txt` and can be found within Blackboard. Test your algorithms on each of the two crash scenarios. It is up to you to determine appropriate design parameters such as learning rate, discount factor, exploration strategy, etc.
- Use the data files for the tracks provided, represented in ASCII. The first line lists the size of the track as an comma delimited pair $\langle rows, cols \rangle$. The rest of the file is a grid of the specified dimensions with one character at each point. The legend for the files is:
  - `S` – This square is on the starting line.
  - `F` – This square is on the finish line.
  - `.` – This square is open racetrack.
  - `#` – This square is off the racetrack (i.e., a wall).

  As a simple example:

  ```
  5,5
  FF###
  ..###
  ..###
  ....S
  ....S
  ```

- Run experiments, keeping track of the number of training iterations and the number of steps the race car needs to take before finding the finish line. Make sure you run at least 10 experiments for each track (preferably more) to get reasonable statistics. Also generate data so you can plot learning curves.
- Write a paper that incorporates the following elements, summarizing the results of your experiments. Make sure you explain the experimental setup, the tuning process, and the final parameters used for each algorithm.
  1. Title and author names
  2. A brief, one paragraph abstract summarizing the results of the experiments
  3. Problem statement, including hypothesis (how do you expect the algorithms to do?)
  4. Description of algorithms implemented
  5. Description of your experimental approach
  6. Presentation of the results of your experiments
  7. A discussion of the behavior of your algorithms, combined with any conclusions you can draw
  8. Summary

    9. References

- Submit your fully documented code with sample runs, results of the runs of each algorithm, and your paper. Your grade will be broken down as follows:
    - Code structure – 10%
    - Code documentation/commenting – 10%
    - Proper functioning of your code, as illustrated by a 5 minute video – 30%
    - Summary paper – 50%