# 605.649 — Introduction to Machine Learning

**Programming Project #4**

## 1 Project Description

For the fourth and final programming assignment, you will implement a reinforcement learner and apply it to the racetrack problem. The racetrack problem is a popular control problem, often used in reinforcement learning assignments. 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 laid out on a Cartesian grid). The variables $v_x = \dot{x}_t$ and $v_y = \dot{y}_t$ represent the $x$ and $y$ components of the car's velocity at time $t$ (note that $\dot{x}$ denotes the first derivative of $x$ with respect to time). The control variables for the car are $a_x = \ddot{x}$ and $a_y = \ddot{y}$, which represent the $x$ and $y$ components of an acceleration vector to be applied at the current time step (note that $\ddot{x}$ denotes the second derivative of $x$ with respect to time). The system is governed by an approximation of the standard laws of kinematics:

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

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 $(v_{x_t}, v_{y_t}) \in [\pm 5, \pm 5]$. 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. This means that you need to write code capable of detecting the vehicle intersecting with the boundary of the track. One common algorithm that can be used for this comes from computer graphics and is called "Bresenham's Algorithm." You can find information on this algorithm at

https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

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, the car has to start over from the beginning, but the "clock" keeps running (i.e., the time trial is still going since the car has not yet crossed the finish line).

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 are required to use the version where crashing stops the car but leaves it in the location it crashed. Time permitting, you are free to compare both variants on all tracks, but this is not a requirement for the assignment.

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.

## 2 Project Requirements

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). Note that this can actually be implemented as a lookup table.

- Implement the Value Iteration algorithm and apply it to the racetrack problem using the model as described above.

- Implement *both* the $Q$-learning and SARSA algorithms, and apply them to the racetrack problem using the racing simulator but no direct knowledge of the model.

- Test your algorithms on the four tracks provided in the data files described below. The files are named L-track.txt, O-track.txt, R-track.txt, and W-track.txt and can be found within Canvas.

- For the L-track, O-track, and W-track, test your algorithms under the condition that a crash results in the vehicle being placed at the nearest point to the crash.

- For the R-track, test your algorithms under the two crash conditions: 1) a crash results in the vehicle starting at the nearest point to the crash with the race continuing, and 2) a crash results in the vehicle moving back to the starting line with the race continuing. In the latter case, the start state closest to the crash should be chosen.

- It is up to you to tune 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 ⟨*rows, cols*⟩. 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 very brief paper that incorporates the following elements, summarizing the results of your experiments. Your paper is required to be at least 5 pages and no more than 10 pages using the JMLR format You can find templates for this format at `http://www.jmlr.org/format/format.html`. The format is also available within Overleaf.

  1. Title and author name
  2. A brief, one paragraph abstract summarizing the results of the experiments
  3. Problem statement, including hypothesis, projecting how you expect each algorithm to perform
  4. Brief description of your experimental approach, including any assumptions made with your algorithms
  5. Presentation of the results of your experiments
  6. A discussion of the behavior of your algorithms, combined with any conclusions you can draw
  7. Conclusion
  8. References (Only required if you use a resource other than the course content.)

- Submit your fully documented code with sample runs, results of the runs of each algorithm, and your paper.

- For the video, the following constitute minimal requirements that must be satisfied:

  - The video is to be no longer than 7 minutes long (note this is different from prior assignments).
  - The video should be provided in mp4 format. Alternatively, it can be uploaded to a streaming service such as YouTube with a link provided.
  - Fast forwarding is permitted through long computational cycles. Fast forwarding is *not permitted* whenever there is a voice-over or when results are being presented.
  - Be sure to provide verbal commentary or explanation on all of the elements you are demonstrating.
  - Demonstrate one sweep of the value iteration algorithm, showing the "$V$" values being updated for the track.
  - Demonstrate the "$Q$" values being updated along one sequence generated during $Q$-learning.
  - Demonstrate the "$Q$" values being updated along one sequence generated during SARSA.
  - Demonstrate the exploration function of your learning algorithm, showing cases where both a current optimal action is selected and where a random action is selected. Note that this is different from demonstrating the nondeterminism of the simulator.
  - Demonstrate the generation of a path of $\langle state, action, state \rangle$ triples of your vehicle at the end of learning on one track for each of the three algorithms. Highlight a point where the nondeterministic response to an action occurs along that path.
  - Demonstrate the "restart" behavior for each of the two scenarios when the car collides with the wall. If you wish, this can be done as part of the path generation demonstration above.
  - Demonstrate your clipping algorithm for detecting when a car collides or intersects with the boundary of the track. If you wish, this can be done as part of demonstrating the "restart" behavior, since restarting involves responding to a collision.

# 3 Project Evaluation

Your grade will be broken down as follows:

- Code structure and commenting – 20%

- Proper functioning of your code, as illustrated by a 7 minute video – 30%

- Summary paper – 50%