RL Project: Navigation Task using Reinforcement Learning and CMACs

Doina Precup

December 21, 1995

Task description

The purpose of the project is to experiment reinforcement learning in for a continuous state space environment. We want to teach an agent how to get from a starting position into a goal region without bumping into the walls or into any obstacles, and without exceeding a specified amount of time. A typical environment is shown in figure 1.

The task is continuous in time and space, but for the purpose of this project, time will be discretized, thus we will consider unit time steps. At each time step, the agent can choose the acceleration on the x and y axis. The motion update equations are:

$$x := x + v_x + 0.5a_x \quad v_x := v_x + a_x$$

$$y := y + v_y + 0.5a_y \quad v_y := v_y + a_y$$

Maximum speed limits on each axis are specified. If these limits are attained, further acceleration will maintain the original speed. Friction is ignored, thus an action always has the same effect on the motion parameters.

In order to avoid obstacles, the agent also has the capability of sensing the environment. We assume that it has a limited view range, and it can sense any obstacles that appear on each of the four axis directions (north, south, east and west, in the absolute coordinate system) within this view range. The walls are also considered obstacles, thus they can be detected in the same way. We shall denote the sensor information by d_n, d_s, d_e, d_w.

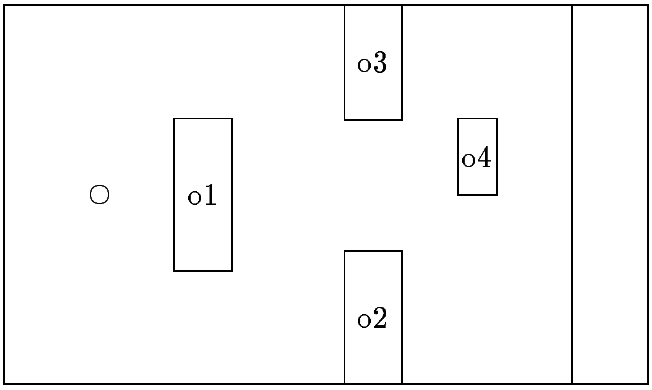

Figure 1: Environement structure

Problem setup

The information that the agent could analyze at any point is its own state (speed and coordinate) and the sensor input. In order to obtain generality for different obstacle arrangements, the position information is discarded. Thus, a situation is described by six parameters:

$$v_x, v_y, d_n, d_s, d_e, d_w$$

There are nine possible actions, corresponding to the possible combinations of acceleration values (+1, 0 and -1) on each axis. The two axis acceleration components can't be treated separately, because the aggregated motion in the case of obstacles depends on both of them.

As stated before, this is not a minimum time problem, the purpose is to meet a certain deadline. Thus, this is a non-discounted, total reward problem. The trial can end in one of the following situations:

- The goal state is reached, which yields a reward of +1
- The agent hit an obstacle, which yields a reward of -1
- The time deadline elapsed, in which case the reward is -0.5

The reward is 0 on all the other time steps.

Implementation

The agent learns state-action values, using sarsa with eligibility traces. In order to handle the continuous variables, function approximation is needed. Thus, we used a set of nine CMAC's, one for each action. Each of the six parameters is divided in three equal sub-intervals, a fourth row and column being added to enable offsetting. Thus, a tiling depending on all the parameters has $4^6 = 4096$ tiles. We used 7 tilings dependent on all the variables, as well as 3 tilings depending on d_n, d_s, v_y and d_e, d_w, v_x respectively. This solution is meant to underline the stronger interdependance between the parameters measured along the same axis. Each tiling was generated at a random offset in the unit interval.

The algorithm is the following:

1. Initialize all the feature-action values and traces: $Q(f, a) = 0, e(f, a) = 0$. The initial values are optimistic for most of the pairs. During training, some of the values will actually go higher than these values. However, for most pairs the values go down. Thus, this setting enables a lot of exploration in the initial stages, decreasing it after accumulating experience.

2. Start trial at random spot, with $v_x = v_y = 0$;

3. Repeat until trial end:

a. Compute the set of current features, F_t, and pick one of the actions that maximize $Q(s_t, a) = \sum_{f_t \in F_t} Q(f_t, a)$. Let this be a_t. Actually, we cinsider for the choice all the actions that are within a small specified distance of the maximum value.

b. Update all the eligibility traces, in two versions: accumulate and replace.

c. Perform the selected action and get the reward r_t. Update the motion parameters and the sensor readings accordingly. Because of the limited sensor capacity, the shape and dimension of the obstacles can't be anticipated, thus we need to take the action in order to get the next state. Step (a) is performed again, giving the best expected action in the next state, a_{t+1}.

d. Update feature-action values:

$$Q(f, a) := Q(f, a) + \frac{\alpha}{ng} e(f, a)(r_t + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)),$$

where ng is the number of grids and the values Q(s,a) are computed as before, by summing the appropriate feature-action values.

Results

The program was run for 100 trials at different values of λ and α, and the process was repeated over 10 runs. In order to provide a fair comparaison, the same starting positions were used for accumulate and replace traces. The environment was 10x10, with allowed speed of the agent between -3 and 3, and view range of 6.

The algorithm converges for values of λ ranging form 0.2 to 0.5. Convergence takes place only for small values of α (0.2 to 0.5). In these cases, the goal is always reached in the end, thus the final average reward is always 1. For these cases, behaviour is highly similar. Thus, the initially desired comparaison of different values isn't too informative. The only difference is a small variation in speed of convergence, the best speed on average being at the values for which figure 2 has been drawn. On other values, convergence hasn't been reached; the values seem to stabilize, then start to oscillate again. For $\lambda = 0$, the system converges to a suboptimal policy, in which time always elapses. Further experiments would be needed, but time is up for the project.

Figure 2 shows how the average reward on 10 trials is varying in time, for the same values of the parameters, with accumulate and replace traces ($\lambda = 0.3, \alpha = 0.5$). Both algorithms yield convergence to a policy in which goal is always reached, but the replace variant seems to induce a more monotonic behaviour, and it reaches the optimal value quicker.

In cases of convergence, the values obtained for different states and actions exhibit indeed generalization capacities. For instance, all the situations with high speed and close obstacle on one direction have much higher values for actions that decelerate on that direction than for any other actions.

Problems

A problem worth mentioning here is that the number of tilings is determinant for convergence. A smaller number of tilings created a hidden state problem, which hindered convergence. Adding more tiles is only one possible solution. More interresting approaches could probably be adopted (Moore and Atkeson, 1996).

Because the position is not included in the state description, another problem has been the early training of the algorithm. In order to create

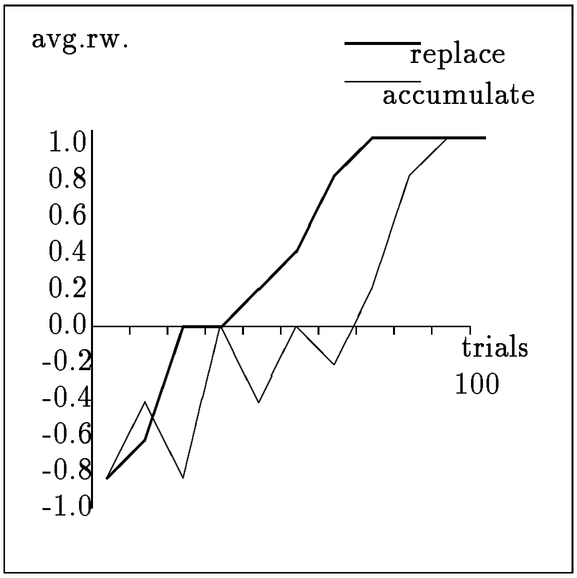

Figure 2: Average rewards variation in time

a bias towards actions that lead to the goal, we had to train first without any obstacles. Afterwards, we started to complicate the setting, adding one obstacle at a time. If all the obstacles are added in the beginning, the algorithm converges to a sub-optimal policy, which yields exit on time elapse at every step. The result from the previous section have been obtained considering that the trials start after the agent has learned to reach the goal in the empty environment, and has trained to convergence only with the first obstacle.

Still an open problem remains the exploration issue. The problem is set up such that the goal remains always the same. Thus, after the agent learns to avoid the obstacles, it will reach the goal all the time, as shown in the picture. Exploration has been ensured in this version by picking randomly an action that is within a specified small distance from the optimal value. Random starts are also determining some exploration. Using an ϵ-soft policy would be another alternative, but we haven't tried it so far.

In order to be able to handle the problem, the agent must be faced with a small environment. In a bigger setting, finer grids are required, which results in a too big state space for the CMAC approach. Another enhancement that would be needed is the improvement of the sensing capacities, with a sonar type of sensor able to detect obstacles on several directions.

It also seems that a problem that hinders convergence is that the agent has many possible actions and in the same situation, the same action can yield to very different outcomes, depending on the obstacle configuration. At the end, I was trying to experiment with just eight simple stepping actions, treated as speed and not as acceleration. I can't report any results on that case yet.

As a conclusion, the project was interresting for me because I bumped into a lot of problems, but I don't think that the results are too great. I will keep experimenting with it, maybe I find some answers to the remaining questions.

Additional bibliography

Singh S., Sutton, R.S., Reinforcement Learning with Replacing Eligibility Traces, to appear.

Moore A., Atkeson, C., The Parti-Game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces, to appear.