RL Project Report

May 11, 2015

Abstract

To use reinforcement learning to make a robot arm learn how to catch a ball through experience. We compare the results by doing this when compared with a deterministic algorithm like RRT (Rapidly exploring random tree).

1 Problem Statement

The aim of this project is to make a robot arm learn to catch a ball using Reinforcement Learning and compare it with conventional techniques or path planning. We use a robot arm model in Matlab and the corresponding dynamics and use an inverse kinematics relation to abstract out the low level control on the angles of the links, and learn to move in the 3D cartesian space. We compare this with a generic Rapidly exploring random tree.

2 Robot Arm Model

In general, the robot arm's model wouldn't matter to the learning algorithm, as the learning part has been abstracted to the cartesian 3D space. The Robot arm we used was the well known PUMA (Programmable Universal Machine for Assembly) arm, which has six degrees of freedon/joints.

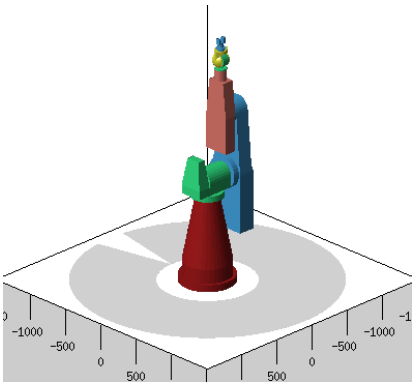

Figure 1: PUMA arm model in Matlab

We took a simple kinematic relation for the robot arms, where a step function is used to model the velocity of each link. Hence, this brings about an uncertainity whether the arm will be going in the direction required in a small time step. Due to the velocity constraint on the linkages (and not x, y, z), if the arm may not even go towards the target position.

The arm is displayed in a MATLAB 3D plot using a solidworks model fo the PUMA arm and converting it into matlab meshes.

RL Project Report

3 Reinforcement Learning solutions-

We solve the problem of catching the ball by formulating it as a MDP. Since the ball can take any position in the defined area and the angles of the links of the arm can attain angles over a continous space, the problem is a continuous state and continuous action MDP. In order to solve this MDP, we take the help of linear function approximation techniques for representing the value functions of the state and use a encoding to represent the various actions onto a finite set.

3.1 Formulating the problem into a RL Framework-

The first task was to choose the features that can be used to encode the states. We decided on a 3 bit binary encoding for the state. If the ball had a x coordinate greater than the x coordinate of the tip of the arm, the first bit was set as 1, else it was set as 0. Similarly the other two bits represented the relative y and z coordinate of the ball with respect to the tip of the arm. Since the aim is for the tip of the arm to reach the ball, we felt such a relative representation is sufficient for the problem.

Even though our arm consists of 6 degrees of freedom, 3 of them are sufficient for the movement of the tip arm to a particular (x,y,z) coordinate. Thus we again

use a 3 bit encoding for the actions corresponding to the change in angle of each link. If the bit corresponding to the first angle is 1, then that link is moved in that action, else it is not. The amount by which it is moved is a scaled version of the distance between the ball and the tip of the arm. The scaling vector was proportional to radial error between the tip of the arm and the ball. Since in our case the ball was dropping in the z plane, the radial error corresponds to the distance between the x and y coordinates. To incorporate real world constraints into the movement of the arm, we also upper bound the maximum change in angle by the maximum velocity that the arm can attain. Thus this prevents the arm from instantaneously changing its position by a large amount.

The reward given to the agent consists of two subrewards. The first subreward is the reward for catching the ball for which a reward of +20 is given. The other subreward depends on the radial error. Penalising the radial error ensures that the arm doesnt move randomly till the ball comes down and tries to move towards the final goal.

3.2 Using SARSA(λ) for control-

We use the linear, gradient descent Sarsa(λ) with binary features, an ϵ-greedy policy for control and replacing traces for controlling the RL agent. The description of the algorithm is given below

Algorithm 1 Linear, gradient-descent Sarsa(λ) with binary features and ϵ-greedy policy

Let $\mathbf{w}$ and $\mathbf{e}$ be vectors with one component for each possible feature
Let $\mathcal{F}_a$, for every possible action a, be a set of feature indices, initially empty
Initialize $\mathbf{w}$ as appropriate for the problem, e.g., $\mathbf{w} = \mathbf{0}$
Repeat (for each episode):
    $\mathbf{e} = \mathbf{0}$
    S, A $\leftarrow$ initial state and action of episode (e.g., ϵ-greedy)
    $\mathcal{F}_A \leftarrow$ set of features present in S, A
    Repeat (for each step of episode):
        For all i $\in \mathcal{F}_A$:
            $e_i \leftarrow 1$ (replacing traces)
        Take action A, observe reward, R, and next state, S'
        $\delta \leftarrow R - \sum_{i \in \mathcal{F}_A} w_i$
        If S' is terminal, then $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{e}$; go to next episode
        For all $a \in A(S')$:
            $\mathcal{F}_a \leftarrow$ set of features present in S', a
            $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} w_i$
        $A' \leftarrow$ new action in S' (e.g., ϵ-greedy)
        $\delta \leftarrow \delta + \gamma Q_{A'}$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{e}$
        $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e}$
        $S \leftarrow S'$
        $A \leftarrow A'$

The episode consists of a set of 10 balls which are dropped from random positions with random velocity and accelerations. The score out of 10, for 50 episodes averaged over 5 trials is given below.

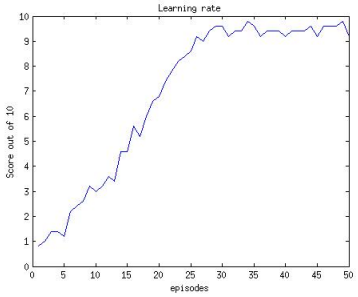

Figure 2: Performance of the agent

3.3 Parallelising the learning of the agent

Even though we have reduced the problem complexity by using function approximation, learning the policy still requires a lot of runs. To speed up the learning process, we made created multiple arms which run in parallel . After each episode, a weighted average of the weights learnt by the various arms is used to initialise the weights for the next episode. The weights are determined by $\frac{r_i}{\sum_{j \in N} r_j}$, where r_i is the number of balls caught by the arm in one episode and N is the number of arms.

A video of the demonstration of the arm has been uploaded at http://youtu.be/VjHS-uqdiEM.