Inverted Pendulum on a Cart Problem

Sergio Azizi

Abstract

Introduction to the problem, MDP's, etc.

1. Problem

A point-mass m is fixed to the end of a pole of length l and is connected to a cart via a free hinge. The cart is restricted to move in one dimension. We want to apply a reinfrocement learning algorithm whose goal it is to balance the pole for as long as possible by accelerating the cart to the left or right direction.

2. Markov Decision Processes

An MDP has five components:

1. A set of states S: This is the set of all possible postions and orientations of our pendulum. We will define each state with:

x-the position along the axis

theta- the angle of the pole with the vertical

xdot the velocity of the pendulum

thetadot its angular velocity

The above variables are are continuous and need to be discretized to be suiatble for our framework. See Discretization for specifics.

2. A set of actions: Our controler will choose one of two actions, accelerate the car to the right (action 1) or to the left (action 2). For simplicity, there is no do-nothing action.

3. set of state transition probabilities: This decribes the probability distribution for each state action pair. In other words, when we are at state s and take an action a, the state transtion probability tells us the likelyhood of getting to a next state s. This set is unknown to us and needs to be estimated from the data (see section on learning a model).

4. A discount factor: A real number between 0 and 1, which devalues future rewards. The intuition is that to maximise the reward we would like to get positive rewards as soon as possible and postpone negative rewards for as long as possible. We chose 0.995.

5. A reward function: in our case, rewards will be a function of the state. In particular, everytime the system fails (i.e. the cart goes out of bounds/the pole falls), a negative reward of -1 is awarded. At every other state we appoint a reward of zero. The algorithm doesnt know yet which states will most likely lead to negative rewards and should be avoided. But thish information can also be estimated from the data.

3. Dynamics of our MDP

We start at some state s0 (x=0 ,xdot=0, theta=0,thetadot=0). Use the current value function and state transtiotn probability for current stte to compute a policy and choose an action according to that policy. Compute the next state s1, by applying the underlying physics of the model.

4. Discretization

Value iteration is an algorithm which works well for MDP that have a finite number of states. We will apply the following discretization on our model:

Divide x into three boxes: -2.4 to -1.5; -1.5 to 1.5; 1.5 to 2.4 Divide Xdot into three boxes: between -0.5 and 0.5, less or above

Therefore we will have 3x3x6x3=162 discrete states. We will define one additional state that corresponds to failure, resulting in 163 discrete states. The function whatstate will map the state (x, xdot) to an integer between 1 and 163, where the 163th state is associated with failure.

5. Learning a model

In most realisitc apllication, we wont explicitly be given the state transition probabilities and rewards, and

need to estimate them from the data. Lets first talk about the Psas:

To start with we can assume Psa to be a uniform distribution over all states In other words, the probability of getting to s for any state action pair is 1/—S—. We then execute the algorithm until the MDP ends (that is the pendulum failed) for a number of trials, and use the gathered experience to derive estimates for the actual state transitioning probabilities:

Psa(s) = number of times we took action a in state s and got to state s/number of times we took action a in sate s

If a particular state action pair has not been tried in any of our trials (reulsting in Psa = 0/0), simply hold on to the uniform distribution. Similarly, we can estimate the reward to be:

R(s) = accumulated reward observed in state s/number of times we were in state s.

6. Training a controller

Earlier we showed that our total payoff is given by R(s0) + yR(s1) +. The goal of our controller is to compute a policy that over time maximizes the expected value of our payoff E[R(s0) + yR(s1) +.]. A policy is any function pi that maps a state s to an action a We define the value function to be the expected sum of discounted rewards upon starting in state s and taking actions according to pi:

Vpi(s) = E[R(s0) + yR(s1) +.—s0=s,pi]

For a fixed policy this can be rearranged into the bellman equation:

Vpi(s0 = R(s) + y*sum(Pspi(s)(s)*Vpi(s)).

The first term is the immediate reward we get, simply from starting in state s. The second term is is the expected sum of future discounted rewards (Pspi(s)(s)*Vpi(s)=Es pspi(s)[vpi(s)]. In a finite-state MDP, we can write down one such equation for Vpi(s) for every states. This results in a set of —S— linear equations with —S— unknowns (ie the value function). Now, the goal of our learning algorithm is reduced to finding the value function that maximises Bellmans equation. Lets define this as the optimal value function V*:

V*(s) = maxpi of Vpi(s). (eq1)

We then define the optimal policy pi*(s) to be:

pi*(s) = maxa sumovers of (Psa(s)*V*(s).(eq2)

One interesting property of pi* is that its the optimal policy for all states s (in other words the same policy pi* attains the maximum in eq1 for all states). So we can use pi* and attain the maximum value function , independent of the initial state that we chose for our MDP.

Solving a system of 163 linear equations is computanally too expensive, instead we will resort to value iteration, a standard reinforcement algorithm which will guess the optimal value function and then repeatedly update it using Bellmans equation. Specifically:

1. For each state s, inititalize V(s) to the value funcion of the previous iteration

2. Repeat until convergence

for every state s, update V(s) = R(s) + maxa of y*sumovers of Psa(s)*V(s)

Whether the algorithm converges to the correct solution doesnt depend on how you initialize V in the first step of the algorithm. However, choosing the of the previous iteration instead of, say a vecotr of all zeros, allows for fastest convergence. (note: Value iteration only works for finite-state MDPs)

7. model blackbox for attaining s'

8. putting it together

Now lets construct the framework for solving the inverted pendulum problem:

1. initaliaze the state

2. initialize the state transtioins probabiliteis to be a uniform distribution. Remember, in order to update this parameter later, we need to keep track of the number of times that we went from state s to state s through action a.

3. Initialize the rewards at each state to zero and keep track of how often we came across each state.

4. Set the discount factor to number close to one. Repeat until there is no learning left to do {

1. compute the policy using eq2