Adan Constanzo

03 Introduction to MDPs

Exercise 1.1: **Self-Play**

Suppose, instead of playing against a random opponent, the reinforcement learning algorithm described above played against itself, with both sides learning. What do you think would happen in this case? Would it learn a different policy for selecting moves?

Since both AI are learning they are applying an evolutionary method to this problem, meaning there must be multiple games for the system to hill-climb a new policy space. However if both states are starting at similar rates there wouldn’t be much of a difference between policy making because the evolutionary method requires the use of searching best outcomes of a higher level/ different play styles. The system can learn but at a slower rate in comparison to a random opponent.

Exercise 1.2: **Symmetries**

Many tic-tac-toe positions appear different but are really the same because of symmetries. How might we amend the learning process described above to take advantage of this? In what ways would this change improve the learning process? Now think again. Suppose the opponent did not take advantage of symmetries. In that case, should we? Is it true, then, that symmetrically equivalent positions should necessarily have the same value?

By taking advantage of the use of dynamic programming, we are already storing optimal policies from previous policies to save computing time for a similar state. By now applying this new form of symmetric state we can then save more computing time and generate a faster learning rate.

For a game where symmetries are possible, the optimal solution would to keep the idea account and make use of this optimization in generating a faster learning rate.

Exercise 1.3: **Greedly Play**

Suppose the reinforcement learning player was greedy, that is, it always played the move that brought it to the position that it rated the best. Might it learn to play better, or worse, than a non greedy player? What problems might occur?

It will learn to player better because if the model is set correctly, a greedy move means that it’s deciding the optimal choice of reward and results into a smatter system of an AI.

Exercise 1.4: **Learning from Exploration**

Suppose learning updates occurred after all moves, including exploratory moves. If the step-size parameter is appropriately reduced over time (but not the tendency to explore), then the state values would converge to a different set of probabilities. What (conceptually) are the two sets of probabilities computed when we do, and when we do not, learn from exploratory moves? Assuming that we do continue to make exploratory moves, which set of probabilities might be better to learn? Which would result in more wins?

In the process of using an exploratory move we focus on learning from experienced states that we might have never had the chance to capture. The two sets of probabilities are from this random exploratory state and the highest value state. The sense of finding an optimal solution from exploring can be random because we do not know the following predictions from exploring but we do know the greediest state.

Exercise 1.5: **Other Improvements**

Can you think of other ways to improve the reinforcement learning player? Can you think of any better way to solve the tic-tac-toe problem as posed?

Let the player see the possible actions the ai is going to achieve and allow the player to counter it so the ai can be directed to a new state.

By applying machine learning techniques such as an artificial neuron network to keep all possible states and predict the greedies actions for a well defined policy.

**Show that your function computes the same value as the equation above. That is, V(S t ) would**

**be the same in either case.**

V(St) <- V(St) + α[V(St+1) - V(St)

V(St) <- V(St) + αV(St + 1) - αV(St)

V(St) <- (1 - α) V(St) + αV(St + 1 )

**What is the Markov Property?**

The Markov property is a state that includes information about all aspects of the past agent-environment interactions that make a difference for the future.

**Exercise 3.1**

Devise three example tasks of your own that fit into the MDP framework, identifying for each its states, actions, and rewards. Make the three examples as different from each other as possible. The framework is abstract and flexible and can be applied in many different ways. Stretch its limits in some way in at least one of your examples.

One example would bet a MDP for a video game AI MOBA ( multiplayer online battle arena ) , where the states would be the heath, kill count, position, actions would be movement/ abilities for attack and skills and rewards as money or increase status in damage, heath, etc.

Another use of a MDP is a DARPA Robot competition where the action would be movement, opening doors, etc, state will be gps location, fire danger, etc and reward will be not falling over and reaching destination based on time.

Finally another simple use of MDP will be to use on house security system where the state would be levels of methane/air , doors/windows open. Actions will be the use of running programs to check levels of molecules or calling the police/fire department and reward is the house being protected.

**Exercise 3.4**

Give a table analogous to that in Example 3.3, but for p(s’, r|s, a). It should have columns for s, a, s’, r, and p(s’, r|s, a), and a| row for every 4-tuple for which p(s,, r|s, a) > 0.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s | a | s’ | r | p(s’, r|s, a) |
| high | search | high | rseach | α |
| high | search | low | rsearch | 1 - α |
| Low | search | high | -3 | 1 - β |
| High | wait | high | rwait | 1 |
| low | wait | low | rwait | 1 |
| Low | recharge | high | 0 | 1 |