AI 2 Course Project
Reinforcement Learning

Head

Dr. Parvin Razzaghi

Teaching Assistant

Ashkan Rahmani Nejad

Written By

Solmaz Mohamadi

Sepide Bahrami

IASBS – winter 2020

Data Structure, Initial and Goal State

The **Data Structure** used in our code is a 2D array of objects. The object is struct() which represents each cell in our game board. Each struct has variables like N, E, S, and W to show the blocks around the cell, R, V and P to set reward, value and policy for each cell, also a number and a value. Both successor_direction and successor_value are used in policy iteration algorithm. On the following we build the 2D matrix from these structures based on the sample.txt in __init__() function of State class.

- **Initial State:** can be any state given, including a 2D matrix. (here sizes are 5 and 25)
- **Goal State:** since it is clear that where the goal cell is, so the agent must walk towards the goal using an optimal path.

Successor Function

Our **Successor** gets a cell as an input and generates all possible actions can be taken from that cell considering the blocks around it. It returns a tuple consisting of the action taken and the next cell to be moved.

Game Algorithms:
Value/Policy Iteration, Q Learning

Value Iteration

In value iteration, we iteratively apply Bellman optimality equation to get the optimal value function. Initially, we start-off with zero value function. In the second iteration, after applying Bellman equation, we end up with 1 for each non-goal cell and 100 for goal cell, which is the result of maximizing over possible actions in a given state. Notice that the 1 resulted from the reward add to the discounted value of successor state, which it was zeros. This is identical to the bellman update in policy evaluation (which will be explained in the following); with the difference being that, we are taking the maximum

over all actions. Once the updates are small enough, we can take the value function obtained as final and estimate the optimal policy corresponding to that.

Policy Iteration

We first initialize a random policy (action) to each cell based on the information we have about the cell (blocks around it), and then this algorithm loops with these two steps until policy convergence:

1. Policy evaluation: Now at first step we have our random initialized policy, we have to answer the question of how good the policy is. Because solving equations to find values is hard to achieve, so we use an estimation of the value for a cell using Bellman equation. We are going to start-off with an initial value function, e.g. zero for all states. We find the value of the specified random policy like this:

   V = action_reward + gamma(previous_value_iteration[next_state])

   Then we update the V for the cell and save the previous one for next evaluation iterations.

2. Policy improvement: Our reason for computing the value function for a policy is to help find better policies. Once we evaluated the random policy, we act greedily with respect to the value function on that policy to give us a new policy.
   Here we use the successor() to generate all possible actions for the cell, then we find the value for each action and now we greedily update our policy to the action which has the greatest value among others.

Once we got that new policy, we evaluate it. That gives us a new value function. Once we have that new value function, we act greedily with respect to it to get a new policy and so on. In this way, the new policy is sure to be an improvement over the previous one and given enough iterations, it will return the optimal policy.

Q Learning

It is a method of reward and punish specially for partially observable environments. It uses a virtual table of states and available actions for each state. When the agent interacts and tests the actions, the results would be recorded in the table. The more agent walks along states the table would be more reliable.

The test results show that the random actions helps the model training; Specially at the beginning. So what we explore at first, giving the agent the agent possibility of random actions, but as it moves and iterates, the probability tend to decrease near to zero. And the agent would take more of maximum value actions.

Finally, after a proper number of iterations, the model is trained and is ready to be tested.

Analysis

Value and Policy Iteration:

1. Is Value iteration path optimal? Why?

   There are some effective elements, such as

   a. Gamma, which determines the importance of future, moves in iteration. Naturally set according to problem size.

   b. Delta, which is compared to Epsilon, a very small number. Delta shows the changes in values of different actions. It should be small enough to satisfy convergence.

   With that being said, in theory with the ideal Gamma and Epsilon, the path should be the optimal. Because all of states of observable and after the training the optimal path will show itself due to rewarding method.

2. Is Policy iteration path optimal? Why?

   Yes - For finite-action MDP's, there exists a finite number of policies. Therefore, policy iteration must converge to the optimal policy in a finite number of steps. The values are increasing with each iteration.

This is important to understand why Policy Iteration will not be stuck at a local maximum.

3. Which one is better? Why?
   Value Iteration – Because policy iteration requires few iterations of the outer loop, although each of these iterations requires an inner (policy evaluation) loop inside. However, Value iteration requires several iteration of its single loop, but each iteration requires no further loop. We can say that policy evaluation part of the policy iteration makes it costly.

Q learning:

1. Is Q Learning path optimal? Why?
   Since it is based on a partially observable problem, there is no guarantee that it would give the optimal path result.

2. What strategy have been used in action choosing? Why?
   The explore and exploit strategy. Since it is a partially observable problem and the agent needs the interaction and take risks more in the beginning of iterations.

Citations

1. https://www.analyticsvidhya.com/blog/2018/09/reinforcement-learning-model-based-planning-dynamic-programming/
2. https://towardsdatascience.com/reinforcement-learning-demystified-solving-mdps-with-dynamic-programming-b52c8093c919
3. https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/