Q-learning in Wumpus World

1 Problem statement

Wumpus world is a toy problem domain used in [3] to introduce the concept of logical agents. Using agents that reason about the world with the tools of formal logic is an intuitive approach that underpinned much of the earlier research in artificial intelligence, however there are issues associated with using formal logic as a basis for reasoning (such as the frame problem). I propose to implement an agent that can learn to reason about its environment. Showing that complex reasoning can be internalized by a learning agent allows us to solve problems that are traditionally hard to describe in the language of formal logic.

2 Related work

Planning in wumpus world like domains would typically be done using SATPLAN [2]. SATPLAN works by transforming the planning problem (find a path through the cave to the gold without entering a square with a live wumpus or a pit), into a satisfiability problem, determining whether the transformed problem is satisfiable, and then transforming the model found by the SAT solver into a plan for the original problem. Q-learning was first introduced in [4], and is a widely used technique for performing reinforcement learning without the need for a model, unlike other reinforcement learning techniques that use the bellman equation, such as value and policy iteration. The task of a q-learner in this scenario is to find a policy which drives the adventurer to explore the cave for the gold without encountering the wumpus or a pit. A similar approach was presented in [1], although it’s focus was more on comparing the trade off between exploitation and exploration.

3 Approach

I implemented a q-learning agent for the wumpus world. The formulation is similar to that given in [3]: The environment consists of a toroidal two-dimensional grid, where each cell can contain either a wumpus, a pit, or a bar of gold. An agent can navigate through the environment by moving in one of four directions. If an agent enters a cell with a live wumpus or a pit, it dies. If it enters a cell adjacent to either of these, it perceives a stench and/or a breeze, respectively. Unlike the formulation given in [3], there are no walls, since the environment is toroidal (meaning each edge wraps around). If an agent attempts to move left at the left-most edge of the cave, for example, it simply moves to the right-most edge (and similar for the top and bottom edges).

The task of the agent is to navigate safely to the gold using these limited perceptions. In order to perform q-learning for this task, a description of the states, actions, and rewards is needed. State, for an agent, is represented as the following tuple

\[(X, Y, \text{ARROWP}, \text{SCREAMP}, \text{HISTORY})\]

where \((X, Y)\) represents the location of the agent, \text{ARROWP} and \text{SCREAMP} represent whether the arrow has been fired, and whether the wumpus has been killed, and \text{HISTORY} is the set of facts it knows about each of cells in the environment so far (e.g. \text{smelly}, \text{breezy}). The actions available to the agent consist of moving or shooting in one of four directions. If the agent shoots in the direction of the wumpus, the wumpus dies and it becomes safe for the agent to enter the wumpus’ cell. The agent is given a reward of 1 for finding the gold, \(-1\) for dying, and zero otherwise.

The general q-learning algorithm I implemented works in the following way:

1. Pick the best action to perform in the current state
2. Perform that action, receive a reward and a new state
3. Update it’s policy given the new reward and state

The policy which provides the best action used in step 1 is embodied in the Q-table, a table that associates state-action pairs with values representing the...
<table>
<thead>
<tr>
<th>Environment</th>
<th>Avg % Successful</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIMA</td>
<td>86.3</td>
<td>15.6</td>
</tr>
<tr>
<td>Random</td>
<td>80.3</td>
<td>13.8</td>
</tr>
<tr>
<td>Random cross</td>
<td>29.3</td>
<td>9.3</td>
</tr>
</tbody>
</table>

Figure 1: Percentage of testing runs successful after training for three test cases: The AIMA canonical environment, a randomly generated environment, and two random environments, one used in training and one in testing.

The quality of the pair. This table is initially uniform, but is updated in step 3 using the following equation:

\[ Q(S_t, a) \leftarrow (1 - \alpha)Q(S_t, a) + \alpha \left[ r + \gamma \max_{a'} Q(S_{t+1}, a') \right] \]

where \( S_t \) is the old state, \( S_{t+1} \) is the new state after performing action \( a \) and receiving reward \( r \), and \( \alpha \) and \( \gamma \) are learning parameters.

4 Evaluation

Figure 1 shows the performance of the agent on three test cases. In all three, the agent was allowed to explore the cave and build up its q-table for a fixed number of runs, where each run ended after the agent found the gold, died, or a time-out expired (calculated to be long enough for the agent to explore each room in the cave multiple times). On each run, the agent’s initial position was randomly chosen from the empty cells. After the initial runs, the agent was tested with the q-table it had produced by counting the percentage of times it was able to find the gold out of 10 testing runs (with random initial positions). The averages and standard deviations are given for 30 repeated trials. Figure 3 shows the performance as a function of the number of training runs allowed before testing. The number of training runs in other figures was fixed at 1000, as to reduce variance in performance.

The first environment is that given in [3]. The second set of environments were generated randomly, as described above, and performance was measured for environments ranging from 4x4 to 9x9. The agent was trained and tested on each environment for the same number of runs as described above, although the time-out was increased proportional to the number of additional states.

5 Discussion

It’s clear that after training, the agent is able to find the gold while avoiding the hazards of the wumpus world with a relatively high percentage, as long as it’s being tested on the same environment it was trained for. The fact that the agent performs comparably on many different randomly generated environments shows that the agent’s performance is con-
sistent across the problem domain. The agent’s performance decreases as the size of the environments increases, although it’s not clear if the trend bottoms out, or continues. One explanation for the decrease in performance, is that in larger environments, it’s harder to find the goal by exploring randomly. This is similar to the problem with the size of the state space, discussed below.

The final result is somewhat surprising, in that the agent performs poorly when the training and testing environments are not the same. One of the original premises of my proposal was that the Q-learner would be able to generalize across multiple environments, but clearly this is not the case. One possible explanation for this is that the state space of the learner is only being explored fairly shallowly around the training environments. The state representation described earlier describes a state space that is exponential in the size of the environment. This presents a problem, because the agent must be aware of it’s perception history in order to learn a policy that generalizes to different environments, but including this history in a naive way leads to an exponential state space. Improving the state space representation in such a way as to allow for the agent to learn generalizable rules would definitely be an interesting area for future work.

References


