Allan Reyes
Machine Learning, OMS CS 7641 (Spring 2017)
Project 4: Markov Decision Processes

Introduction

In this assignment, I analyzed two interesting Markov Decision Processes (MDPs) from the OpenAI Gym Toy Text environment[1]. Both are solved using exact solution methods: value iteration and policy iteration. Both MDPs are then applied with Q-learning.

Problems

Frozen Lake[2]

This first MDP involves controlling an agent in a 2D grid world. The agent starts in the top left (S, blue) and must navigate to the goal in the bottom right (G, green). However, the actions are nondeterministic; moving forward has a 33% probability of going to the left and a 33% probability of going to the right. Moreover, "falling" on a hole (H, red) results as a loss. Additionally, the agent only has 100 "turns" to make it to the goal. The agent receives a reward of 1.0 for reaching the goal, and 0.0 for all other terminal states. OpenAI Gym's environment, "FrozenLake-v0" represents the grid in ASCII:

```
S F F F
F H F H
F F F H
H F F G
```

Figure 1: Grid world for Frozen Lake

I chose this problem because the MDP is simple and the state space is relatively small, encoded in only 16 observations corresponding to each position in the grid world. However, the MDP itself is interesting because while an exact solution can be determined, the goal cannot always be reached, due to a finite horizon of 100 turns. I also found this interesting because of the technology behind OpenAI Gym as well; this particular problem is well benchmarked on the website. Lastly, the infrastructure was easily adaptable to the 8x8 grid world variant[3]:

```
S F F F F F F F
F F F F F F F F
F F F H F F F F
F F F F F H F F
F F F H F F F F
F H H F F F H F
F H F F H F H F
F F F H F F F G
```

Figure 2: Grid world for the 8x8 variant of Frozen Lake

[1] https://gym.openai.com/envs#toy_text
[2] https://gym.openai.com/envs/FrozenLake-v0
[3] https://gym.openai.com/envs/FrozenLake8x8-v0

Taxi

The second MDP involves controlling a “taxi” agent in a 2D grid world. I chose this problem due to the larger state space (30x larger), and because of the literature in which it was presented (Dietterich 2000). I was anxious to upload my solution to OpenAI; unfortunately, the version mismatch between the gym library (Taxi-v2) was incompatible with the server’s version (Taxi-v1).

As a contrast to the previous example, this MDP involved randomized starting conditions. The agent starts randomly on the board, and its goal is to pick up a passenger from a randomly selected location of four locations and drop them off on one. The reward structure is also notably different and more complex than the previous MDP, where successful dropoffs are awarded +20, -1 is deducted for every timestep, and a (massive) 10 point penalty for illegal pick-up/drop-off actions. The state space is notably larger, where the learner must store 500 values at each node, because there are 500 states; 25 locations, 4 possible destinations for the passenger, and 5 possible current locations for the passenger (the four special locations and inside the taxi itself).[4]

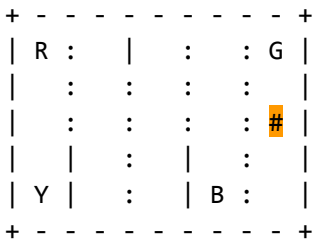

Figure 3: Grid world for Taxi

Exact Solutions

Frozen Lake

Using both value and policy iteration, optimal policies were computed for both the 4x4 and 8x8 grids. Notably, there was a *small* discrepancy on the 8x8 grid due to randomly started policies and terminal states; the value iteration implementation assumes starting with all *left* actions, while the policy iteration implementation populates a randomly seeded policy. The optimal policy for the 4x4 grid is shown below:

[4] Verbatim from Dietterich 2000 (see Bibliography).

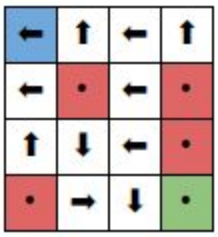

Figure 3: Optimal policy for the 4x4 Frozen Lake

For the 4x4 Frozen Lake, value iteration converged after **27** iterations at an average of **171ms** execution time (across 10 trials), while policy iteration converged after only **4** iterations at an average of **721ms**.

The 8x8 variant of Frozen Lake was also exactly solved:

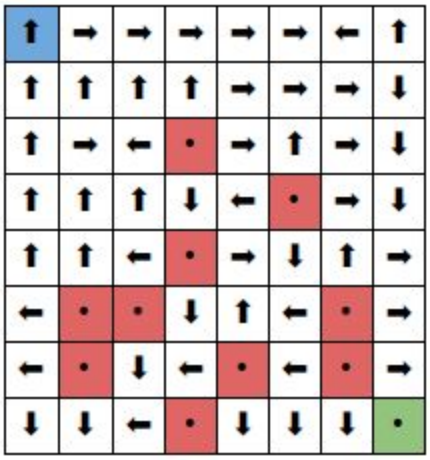

Figure 4: Optimal policy for the 8x8 Frozen Lake

Value iteration converged after **27** iterations at an average execution time of **180ms**, while policy iteration converged after **7** iterations at an average time of **1.1s**.

Taxi

Optimal policies were also exactly solved for the taxi problem, despite the 30-fold increase in the state space. Due to the randomized starting positions of the passengers, however, the policy cannot be simply represented like the Frozen Lake optimal policies, e.g. a passenger in the “R” position might yield a (3, 3) policy of “left,” but yield a totally different policy if the passenger was picked up, or if the passenger started in a different location.

Value iteration converged after **95** iterations in **1.1s**, while policy iteration converged in **16** iterations in **6s**.

Overview

The two problem variants and the results and performance of the MDP exact solution methods are shown below.

Problem	Algorithm	# States	Time (ms)	Iterations	Time/Iter.
FrozenLake-v0	Value Iteration	16	171	27	6.3
FrozenLake-v0	Policy Iteration	16	721	4	180.3
FrozenLake8x8-v0	Value Iteration	64	180	27	6.7
FrozenLake8x8-v0	Policy Iteration	64	1,129	7	161.3
Taxi-v2	Value Iteration	500	3,103	95	32.7
Taxi-v2	Policy Iteration	500	6,012	16	375.8

Figure 5: Aggregate results of value and policy iteration

In all cases, value iteration converged faster in wall-clock time, but policy iteration always converged in less iterations. This trade-off is similar to that of hill-climbing optimizations vs. MIMIC[5]; should the cost to evaluate episodes have been higher (e.g. a physical or manual process), policy iteration offers more benefit. As all the MDPs discussed are of finite-state, they're guaranteed to converge in both value and policy iteration. Given the three example MDPs, the increase in number of states increases the number of iterations for both value iteration and policy iteration. Given a larger action-state space (than 500), it's likely that policy iteration would have outperformed value iteration in wall-clock time to convergence, as well, due to the trade-offs of solving for the Bellman equations vs. estimation of the value function.

Q-Learning

General

Q-learning is a model-free reinforcement learning algorithm that can be used iteratively across episodes to learn the optimal policy in MDPs. The core algorithm is similar to the value iteration update, and defined by the update rule:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

This concise, yet powerful algorithm was implemented in Python to apply against the two previous MDPs. The main hyperparameters to tune are the *learning rate (α, alpha)* and the *discount (ɣ, gamma)*. In both MDPs, the parameters were manually tuned until the Q-learner converged onto the optimal policies (within a certain threshold of iterations). In terms of the OpenAI Gym standards, which are opaque to the internally-represented policies, convergence (or success of "solving" the problem) was measured by the consecutive, running average reward.

[5] Mutual Information Maximization for Input Clustering

Frozen Lake

The first Q-learner to succeed according to OpenAI Gym's standards converged after 3,056 episodes. The "replay" and interactive data charts are available here[6]. After manual tuning, the Q-learner was found to converge at faster rates (~1,000 episodes) at a learning rate of **0.75** and a discount of **0.95**. The results are shown below:

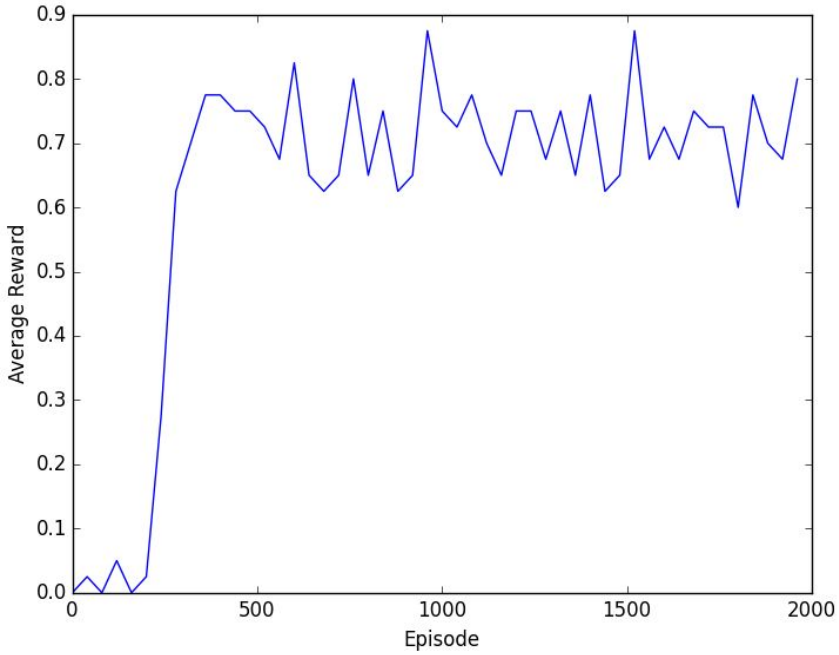

Figure 6: Average reward (smoothed over 40-episode chunks)

Interestingly, the performance of the algorithm was not very sensitive to the hyperparameters. Specifically, the importance of exploration strategy and selection is markedly less important due to the high level of nondeterminism in the actions; any action taken in a non-wall grid square successfully transitions to the intended grid square only 33% of the time. In a way, the underlying model, inaccessible to the Q-learner, consistently promotes exploration due to probability.

An interesting property of the Frozen Lake model is that reward and utility (long-term) are essentially equivalent. No intermediate actions, short of hitting the goal, produce any rewards or penalties. However, the finite horizon does contribute to making the optimal select a path that will lead an agent the goal as fast as possible. In this regard, benefits of Q-learning and the concept of delayed rewards (e.g. taking a short-term penalty for longer-term reward) are less felt.

[6] https://gym.openai.com/evaluations/eval_aem3OASxRWmxPI9SGhuiLQ

The Q-learner is powerful, but “naive” in that it contains no domain knowledge of the underlying model. I was interested in seeing how the addition of domain knowledge and a partial model might affect performance and convergence to an optimal policy, and how much “faster” I could have made it. Adding some production rules to produce a “Semi-informed Q-learner,” such as more strongly favoring an “opposite” direction than would lead to holes was shown to increase the convergence time by a factor of 10. This algorithm was deployed to OpenAI Gym and converged after only 351 episodes. The “replay” and interactive data charts are available here[7]. A comparative study of the two different learners is shown below:

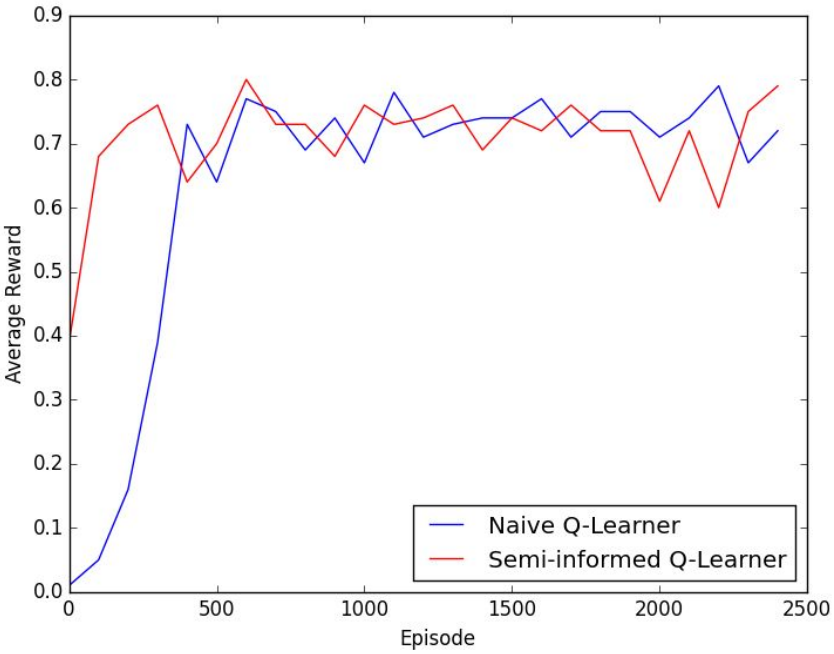

Figure 7: Average reward of a Naive Q-learner vs. a Semi-informed Q-learner

Taxi

Unfortunately, a version mismatch between the library (Taxi-v2) and the OpenAI Gym repository (Taxi-v1) prevented uploading of results and benchmarking against other algorithms. Nonetheless, the same methods of comparing optimal policies and running average reward were used to assess convergence. The hyperparameters were tuned manually, although not necessarily optimized.

Surprisingly, despite the 30x larger action-state space, it was not difficult to achieve a faster optimal policy learning convergence (by number of episodes) than the frozen lake. Notably, the determinism of the taxi cab agent’s actions allowed the Q-learner to essentially learn faster and

[7] https://gym.openai.com/evaluations/eval_hcAYF1otQuuc2ZNVGb3m3Q

exploit optimal actions when desired, versus having to forcibly “explore” due probabilistic nondeterminism. Moreover, the random starting positions allows the agent to have better coverage over the grid world area during early steps; this is in stark contrast to the Frozen Lake scenario, where the agent *always* started on the top left, and rarely landed on regions like the top right.

The results of a convergent trial are shown below, with an learning rate of **0.85** and discount of **0.99**.

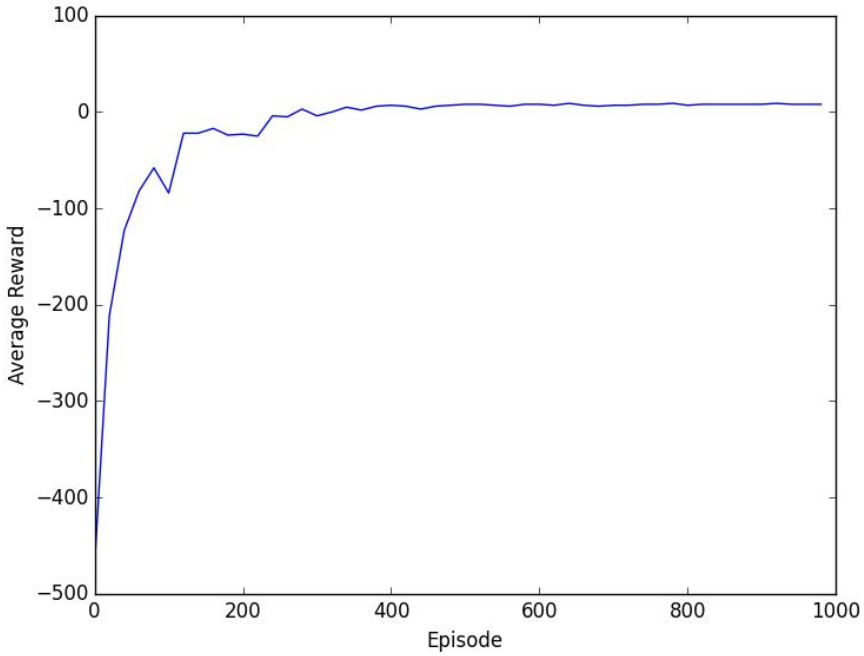

Figure 8: Average reward of the Taxi Q-learner

Noticeably, the initial rewards are *well* below zero. This is due to the exploration strategies where the agent might randomly try to illegally pick-up or drop-off a passenger where there is none, incurring a penalty. Needless to say, the Q-learner learns very quickly to minimize those actions. The “jaggedness” observed happens with spikes in increasing reward when the agent successfully drops off a passenger, reaps the reward, and learns and attributes the action to the Q matrix. Once those actions are learned, the Q-learner gradually gets better as the agent learns the policies to produce the shortest paths to and from correct pick-up and drop-off locations.

The Taxi Q-learner could have gained the most benefit from a semi-informed model approach. Specifically, encoding production rules to “only pick up if there is no passenger in the car” and “only drop off if there is a passenger in the car” could have greatly mitigated the negative rewards. Unfortunately, the discrete state space of OpenAI Gym’s library was opaque, and it was not clear which states included passengers inside the car. Interestingly, the actual

Q-learning agent was truly model-free; all it observed for states was an integer from 1 to 500. The fact that it was able to learn how to pick up and drop off passengers from that singular input is demonstrative of how powerful Q-learning and reinforcement learning can be!

Bibliography

1. OpenAI Gym, http://gym.openai.com
2. NumPy, http://www.numpy.org
3. Matplotlib, http://matplotlib.org
4. Dietterich, T. G. (2000). Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303. http://doi.org/10.1613/jair.639
5. Mitchell, T. (1997). Machine Learning. McGraw Hill. p. 2. ISBN 0-07-042807-7.