CS 533: Intelligent Agents and Decision Making

Homework #5 Report

Michael Lam

Due: May 23, 2016

This assignment explores policy simulation of a "parking" MDP and implements a reinforcemnet learning agent for this environment.

1 Part I

We designed our MDP as suggested, only we further factorize location L into column $C \in \{A, B\}$ and row $R \in \{1, 2, \ldots, n\}$ where n is the number of parking rows. Therefore our state is a 4-tuple (C, R, O, P), which just makes the location more explicit.

We experimented with two sets of parameters. We made the following assumptions for the first one ("MDP1"), which we will call the "reasonable driver":

- The reward for states without being parked is -1 to discourage from driving too much.
- The reward for parking as a function of being in row $r \in \{2, 3, \ldots, n\}$ (note: not in handicap spot) is $(n-(r-1))/(n-1) \cdot F$, where $F = 10$ is an adjustable reward factor. It is more desirable to park nearer to the store.
- The reward for parking in the handicap spots is -100. The assumption here is that the driver really doesn't want to risk a ticket.
- The reward for parking in an occupied spot, i.e. collision, is -10000. Collisions are really, really bad.
- The reward for the terminal state is 1. The terminal state also has no transitions so it's the last state.
- After a DRIVE action, if the next spot is not a handicap spot, the probability that the next spot is occupied by another car is a linear function of the row r: $(n-(r-1))/(n-1)$. As the row goes further away from the store, the probability of being occupied is decreased.
- After a DRIVE action, if the next spot is a handicap spot, the probability that the next spot is occupied by another car is fixed at 0.001, which is a relatively small probability.

The second one ("MDP2"), which we will denote as "ignorant driver", is the same as above except for the following:

- The reward for parking in a handicap spot is 100. The driver doesn't care about getting a ticket and just wants a spot, especially at the front of the store!

This one modification is good enough to test the algorithms and policies. It would make sense that the policies of the reasonable driver and ignorant driver would give different average total rewards. We will test this.

In all the experiments, we set number of rows $n = 10$. We set the discount factor $\beta = 0.99$, which is close to 1 to let the behavior be mainly influenced by the MDP dynamics and reward.

We use policy iteration to get the exact optimal policy, which will be checked against the manual policies specified in Part II and the policy learned by a reinforcement learning agent in Part III.

2 Part II

2.1 Design

Our custom policy is as follows:

> The policy selects DRIVES whenever O is TRUE. When O is FALSE and the current spot is a handicap, the policy selects PARK with probability p_h and otherwise DRIVES with probability $1-p_h$. Otherwise when O is FALSE and the current spot is not a handicap, the policy selects PARK with probability p and otherwise DRIVES with probability $1-p$.

Essentially, we add another parameter called the handicap parking probability p_h. Note that $p_h = 0$ means the agent will never park in the handicap spot and $p_h = 1$ means the agent will always park in the handicap spot when it is not occupied. We experiment with $p_h = 0$ and $p_h = 1$ in our experiments.

Besides our custom policy, we can also compute the optimal policy of the MDPs. We use the policy iteration algorithm from the prior assignment to compute the optimal policy. This will be useful to compare against the other policies as well as use in Part III.

2.2 Results/Experiments/Discussion

We plot the table of average total reward from simulating the two MDPs on various policies in Table 1. The random policy performs the worst, which makes sense since the policy does not handle collisions. The safe random policy (where it always DRIVES on O is TRUE) performs better as it never causes collisions. However, as it still uses randomness to determine whether to park, it can still be better by reasoning the current row and whether it's a handicap spot.

Table 1: Average Total Reward of Various Policies and MDPs

Policy	MDP1	MDP2
Random	-3932.09	-3712.91
Safe Random	10.28	26.23
Custom $p_h = 0$	21.811	20.512
Custom $p_h = 1$	-50.975	64.576
Optimal	78.484	89.06

Our custom policy handles the handicap spot with a probability. By setting $p_h = 0$, the agent will never park at a handicap spot. By setting $p_h = 1$, the agent will always park at a hanicap spot if it is not occupied. For MDP1, we see that $p_h = 0$ gives better results than the safe random policy and we see that $p_h = 1$ performs poorly. This makes sense since in MDP1, there is negative reward for parking at a handicap spot so avoiding it is an improvement over the safe random policy, but forcing to park at a handicap spot yields very poor results from the penalty of parking in a handicap spot. For MDP2, we see that $p_h = 1$ gives better results than the safe random policy and we see that $p_h = 0$ gives poorer results than the safe random policy but it is not drastically bad. This makes sense since in MDP2, there is high reward for parking in the handicap spot so forcing to park in the handicap spot is good, while avoiding the handicap spot is still okay but does not yield as high of a reward.

Finally, we see that all of our manual policies perform worse than the optimal policy. This makes sense since none of our manual policies reasoned about the row the agent is in; we can do better. In Part III, we will construct a reinforcement learning agent which will hopefully learn the optimal policy.

3 Part III

3.1 Design

We implemented a model-free reinforcement learning agent for the environment using Q-learning. It stores a Q-table of states and actions.

For the explore/exploit policy, we use ϵ-greedy: with ϵ probability, choose a random action for exploration, otherwise exploit using the current Q-table. We investigate various ϵ settings in our experiments.

To speed up learning, we use reverse updates. We store the trajectory of a learning trial and do the Q-updates in reverse order. We will also explore different learning rates α in our experiments.

3.2 Experiments/Results

We run $N = 10,000$ learning trials to perform Q-updates, then run $1,000$ trials without learning to get the average total reward. We repeat this 15 times, which we denote as the

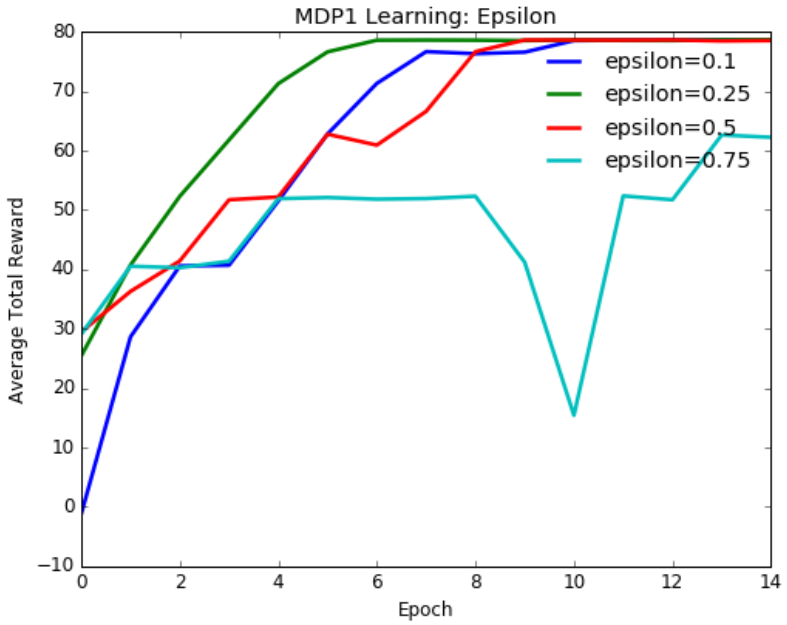

Figure 1: Plot of average total reward as a function of epochs for different ϵ values using MDP1 and fixing learning rate $\alpha = 0.01$.

number of epochs (epoch 0 is the result after running the first $N = 10,000$ learning trials and $1,000$ trials without learning). We experimented by varying two sets of parameters: learning rate α and explore/exploit parameter ϵ.

We experimented with various fixed ϵ parameters: $\epsilon = 0.1, 0.25, 0.5, 0.75$. The graphs are plotted for MDP1 and MDP2 in Figures 1 and 2. We also experimented with various fixed learning rate α parameters: $\alpha = 0.001, 0.01, 0.1, 1.0$. The graphs are plotted for MDP1 and MDP2 in Figures 3 and 4.

3.3 Discussion

First, we analyze what changing ϵ does. For MDP1, we see in Figure 1 that for $\epsilon = 0.1, 0.25, 0.5$ the average total reward eventually converges at epoch 14 to the average total reward as reported from using the optimal policy. For $\epsilon = 0.75$, the behavior is a bit erratic and would probably converge to the average total reward using the optimal policy had there been more epochs. We also see a similar behavior for MDP2 in Figure 2. The erratic behavior and slow convergence for $\epsilon = 0.75$ makes sense since a higher ϵ means higher probability of exploration constantly. Higher probability of exploration is good at the beginning when we have no knowledge. However, as we gain more knowledge we should start exploiting it rather than exploring. The experiment confirms that intuition. Overall, the plot is also a good check that the Q-learning reinforcement learning algorithm is indeed learning the optimal policy through Q-tables

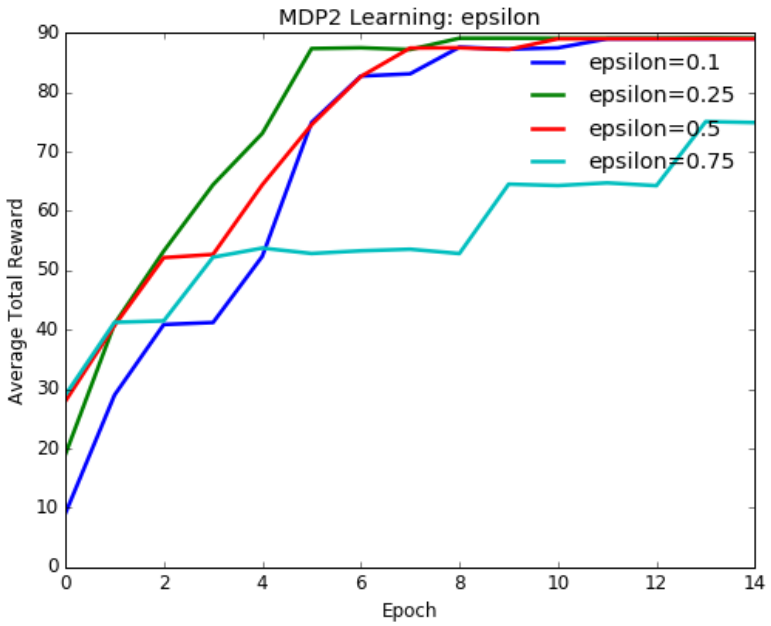

Figure 2: Plot of average total reward as a function of epochs for different ϵ values using MDP2 and fixing learning rate $\alpha = 0.01$.

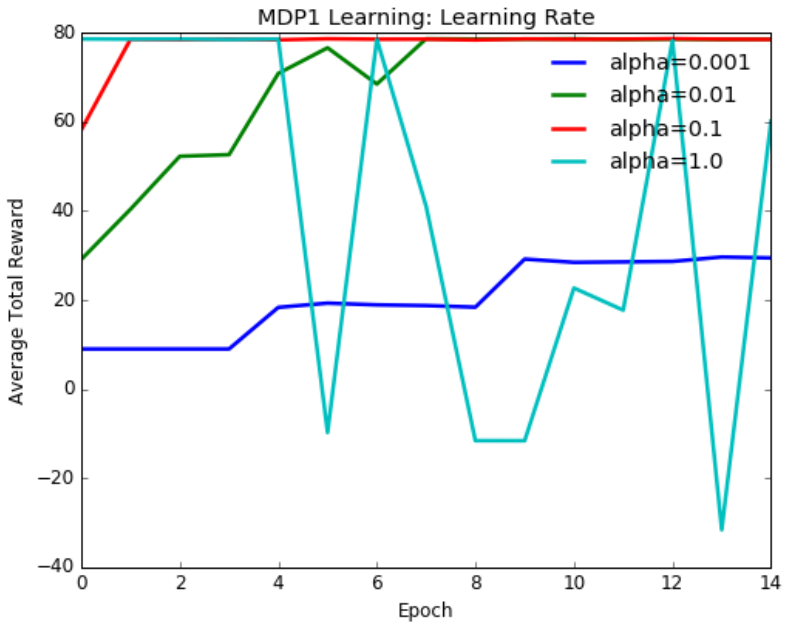

Figure 3: Plot of average total reward as a function of epochs for different learning rate α values using MDP1 and fixing $\epsilon = 0.1$.

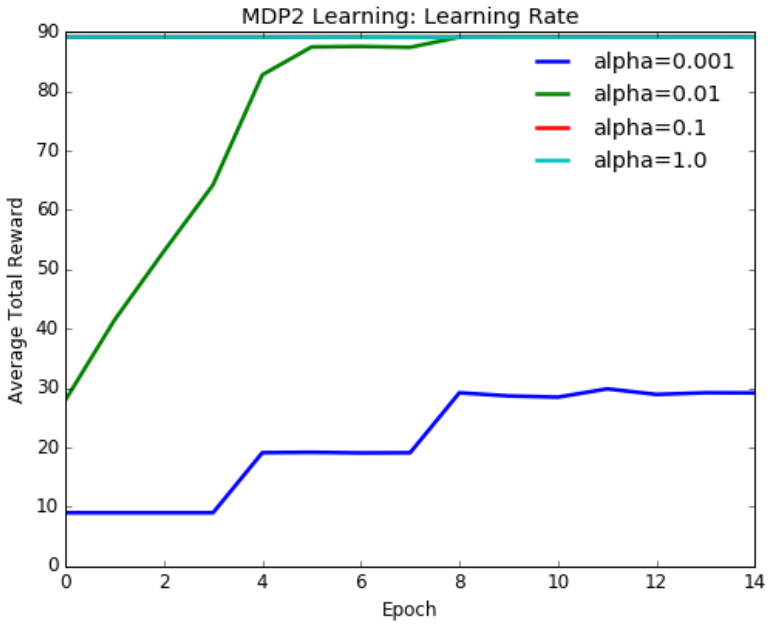

Figure 4: Plot of average total reward as a function of epochs for different learning rate α values using MDP2 and fixing $\epsilon = 0.1$.

since the average total reward converges to that of the optimal policy.

Next, we analyze what changing learning rate α does. For MDP1, we see in Figure 3 that for $\alpha = 0.01, 0.1$ the average total reward eventually converges at epoch 14 to the average total reward as reported from using the optimal policy. For $\alpha = 0.001$, the average total reward grows slowly and will probably converge to the average total reward using the optimal policy had there been more epochs. For $\alpha = 1.0$, the behavior is eratic: it converges at first, but then oscillates to lower values. Overall, the lower the α the more stable the training and the more slowly until convergence. The higher the α, the faster the training but a high enough α will cause instability. This makes sense and is consistent with learning algorithms like gradient descent.

For MDP2, the analysis of α is slightly different as shown in Figure 4. We see that for $\alpha = 0.01, 0.1, 1.0$ the average total reward eventually converges at epoch 14 to the average total reward as reported from using the optimal policy. The difference is that $\alpha = 1.0$ is no longer unstable and in fact converges. This probably makes sense for MDP2 since the optimal behavior would be to park at the handicap spots since it has the highest reward. This rule of always choosing the handicap spot seems easier to learn. Finally for $\alpha = 0.001$, the average total reward grows slowly and will probably converge to the average total reward using the optimal policy had there been more epochs. This is the same as before.

4 Code

Run the code as follows:

```
python setup.py
source venv/bin/activate
python main.py
deactivate
```

The setup script creates a Python virtual environment and main.py runs the script for all parts of the assignment. The setup script is not necessary if numpy is already installed on the system.