# CS181 Assignment 5: Markov Decision Processes and Reinforcement Learning

Ina (Weilin) Chen and Kathy Lin

April 26th, 2013

1. (a) Under a decision-theoretic strategy, we want to choose the policy that gives the greatest expected reward.

   $$\pi^*(s) = argmax_a \sum_{s'} p(s'|s,a)R(s',a)$$

   (b) This proposal makes a lot of intuitive sense because we want to reward the AI for getting the highest possible value below the current score. However, it does not take into account the fact that some regions are harder to hit than others, so if the current score is 61, it would rather try for 60 points and then 1 instead of a more manageable sequence of smaller value throws. Thus, it is more likely to make better decisions when the score is low. Also, this strategy does not consider that some values are not possible. For example, on the small board with only 4 wedges, it is not possible to get 7 points. So if the current score is 19, this strategy would aim for 12-6-1. However, a shorter way to victory is 10-9.

2. (a) The states are all the possible scores (from 0 to the start score). The actions are all the possible locations we can aim at.

   (b) The reward function rewards the algorith for scoring high numbers that are equal to or below the current score. Thus, the reward function grants +1 reward for each point that the program is able to decrease its score by. The discount factor is applied when calculating future rewards so that the algorithm get less and less reward as the game goes on longer.

   (c) See code.

   (d) We want to choose an infinite horizon because it is possible for the darts game to go on forever (for example, if it happens to score 0 every time).

   (e) The optimal policy tends to aim for a region that gives it the highest score that is less than or equal to the current score, because this provides the highest reward possible. When there is a choice between multiple regions that would give the highest reward possible, the policy chooses the safest spot so that if it misses, it will still hit a region that gives a score less than the current score.

(f) As gamma increases, the policy becomes less and less greedy because the reward for prolonging the game does not deteriorate as quickly. For some scores, it no longer chooses the region with the highest possible number of points. When gamma becomes greater than 1, the algorithm actually chooses to miss as much as possible to make the game go on as long as possible because the rewards increase as the game goes on.

3. (a) The optimal policy $\pi^*$ desired is:

$$\pi^*(s) = \underset{a\in A}{argmax}[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')]$$

where $V^*(s)$ is satisfies the Bellman

$$V^*(s) = R(s, \pi^{old}(s)) + \gamma \sum_{s'} P(s'|s, \pi^{old}(s))V^*(s')$$

When we run policy iteration, we first update the values for each state $V^{old}(s)$ using the current policy $\pi^{old}(s)$.

$$V(s) = R(s, \pi^{old}(s)) + \gamma \sum_{s'} P(s'|s, \pi^{old}(s))V(s')$$

Using these values, the new policy $\pi^{new}(s)$ is updated as

$$\pi^{new}(s) = \underset{a\in A}{argmax}(R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s))$$

We terminate when we stop improving, or $\pi^{new}(s) = \pi^{old}(s)$. When $\pi^{new}(s) = \pi^{old}(s)$, the $V(s)$'s calculated from the two policies are also equal. That is

$$\begin{aligned}
V^{old}(s) &= R(s, \pi^{new}(s)) + \gamma \sum_{s'} P(s'|s, \pi^{new}(s))V^{new}(s') \\
&= R(s, \pi^{old}(s)) + \gamma \sum_{s'} P(s'|s, \pi^{old}(s))V^{old}(s') \\
&= V^{new}(s) \\
&= \underset{a\in A}{max}[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^{old}(s')], \forall s
\end{aligned}$$

This equation is exactly the Bellman equations we are trying satisfy. Thus, $V^{old}(s) = V^{new}(s)$ is the optimal values $V^*(s)$. From this we get that $\pi^{new}(s) = \pi^{old}(s) = \pi^*(s)$ and thus we have reached the stationary optimal policy.

(b) Starting with the initial policy $\pi^0$, we update the policy to $\pi^1$. Suppose this policy improves some action for state $\hat{s}$. So $V^{\pi^0}(\hat{s}) \leq Q^{\pi^0}(\hat{s}, Q^{\pi^1})$ where $Q^{\pi^0}(\hat{s}, Q^{\pi^1}) = R(\hat{s}, \pi^1(\hat{s})) + \gamma \sum_{s'} P(s'|\hat{s}, \pi^1(\hat{s}) V^{\pi^1}(s')$. Then,

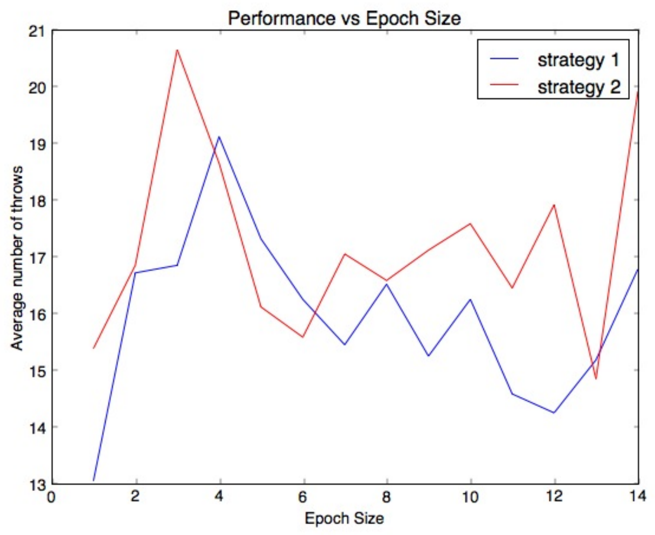
$$\begin{aligned}
V^{\pi^0}(\hat{s}) &\leq Q^{\pi^0}(\hat{s}, Q^{\pi^1}) \\
&= R(\hat{s}, \pi^1(\hat{s})) + \gamma \sum_{s'} P(s'|\hat{s}, \pi^1(\hat{s}) V^{\pi^0}(s') \\
&\leq R(\hat{s}, \pi^1(\hat{s})) + \gamma \sum_{s'} P(s'|\hat{s}, \pi^1(\hat{s}) Q^{\pi^0}(s', \pi^1(s')) \\
&= R(\hat{s}, \pi^1(\hat{s})) + \gamma \sum_{s'} P(s'|\hat{s}, \pi^1(\hat{s})(R(s', \pi^1(s')) \\
&+ \gamma \sum_{s''} P(s''|s', \pi^1(s') V^{\pi^0}(s'')) \\
&\leq R(\hat{s}, \pi^1(\hat{s})) + \gamma \sum_{s'} P(s'|\hat{s}, \pi^1(\hat{s})(R(s', \pi^1(s')) \\
&+ \gamma \sum_{s''} P(s''|s', \pi^1(s') Q^{\pi^0}(s'', \pi^1(s''))) \\
&... \\
&= V^{\pi^1}(\hat{s})
\end{aligned}$$

As we continue to substitute for the $V^{\pi^0}(\hat{s})$ and $Q^{\pi^0}(\hat{s})$ terms, we are carrying out the computation of $V^{\pi^1}(\hat{s})$. Since the choice of $\hat{s}$ is arbitrary, we can generalize this result to all state:

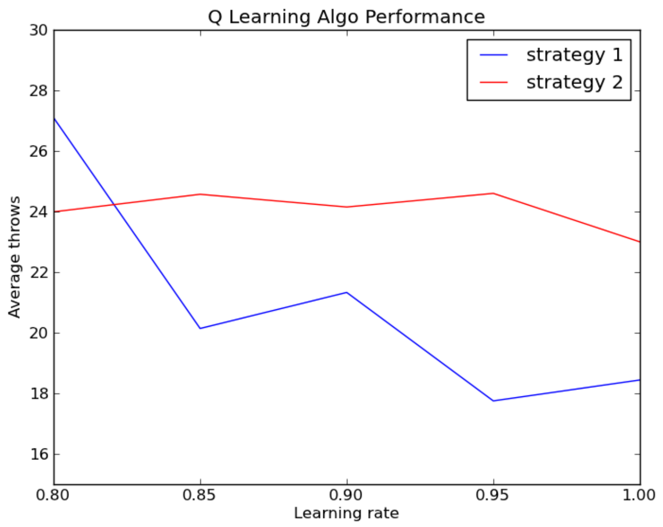
$$V^{\pi^0}(s) \leq V^{\pi^1}(s)$$

(c) We know from (a) that the terminating policy will be the optimal stationary policy $\pi^*$. We also know that $V^{new}(s) = V^{old}(s)$ at the termination. We know from (b) that at each iteration, $V(s), \forall s$ does not decrease, and since we terminate when equal, the $V(s)$ values continues to increase monotonically for at least one $s$ at each iteration. We know that there are a finite number of policies. Thus the algorithm must terminate at the optimal stationary policy with a finite number of iterations.

4. (a) The first exploitation strategy is epsilon greedy (explore with probability (epsilon/num interations). Under this strategy, the algorithm explores less as the game goes on. For the second strategy, the algorithm exlores for a set number of turns before exploiting. Here is the graph for all the epoch sizes between 1 and 15. Both strategies seem to perform the best when the epoch size is between 6 and 12.



(b) Using the same exploration/exploitation strategies as part a, but applied to the Q learning algorithm, we arrived at the following graph for number of throws for each strategy. We used 100 games for the learning algorithm and tested the average performance over 100 games. We found that this range of learning rates performed the best.



We see that as learning rate increase, the learned Q model gets better. This effect is best observed in strategy 1.

(c) The model based algorithm at optimal epoch size performed relatively better than the model free algorithm at the optimal learning rate. The model based algorithm performs better because it is trying to learn about the world rather than just learning how to act as the model free algorithm is doing. In addition, the model based algorithm is able to better take advantage of the exploration/exploitation properties to first learn about the world then act upon the learned knowledge. However, the model based algorithm is much more computationaly costly (owing to its need to compute all the needed information about the world) and thus takes much longer to run than the faster model free algorithm.