CSE 250A. Assignment 8

Hao-en Sung (A53204772)
wrangle1005@gmail.com

November 21, 2016

8.1 Policy improvement

(a) Policy evaluation

Sol. From the lecture, I know that I can solve $V^\pi(s)$ through

$$\begin{aligned}[I-\gamma P^\pi]V^\pi &= R \\ V^\pi &= [I-\gamma P^\pi]^{-1}R.\end{aligned}$$

Thus, I have

$$\begin{aligned}V^\pi &= \left(\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} - \frac{3}{4}\begin{bmatrix}\frac{1}{3}&\frac{2}{3}&0\\\frac{2}{3}&\frac{1}{3}&0\\0&\frac{2}{3}&\frac{1}{3}\end{bmatrix}\right)^{-1}\begin{bmatrix}24\\-6\\12\end{bmatrix} \\ &= \begin{bmatrix}\frac{12}{5}&\frac{8}{5}&0\\\frac{8}{5}&\frac{12}{5}&0\\\frac{16}{15}&\frac{8}{5}&\frac{4}{3}\end{bmatrix}\begin{bmatrix}24\\-6\\12\end{bmatrix} \\ &= \begin{bmatrix}48\\24\\32\end{bmatrix}\end{aligned}$$

□

(b) Greedy policy

Sol. I first calculate the value of $\sum_{s'} P(s'|s,a)V^\pi(s')$ as follows.

$$\begin{aligned}\sum_{s'} P(s'|1,\downarrow)V^\pi(s') &= \frac{1}{3}\cdot 48 + 0\cdot 24 + \frac{2}{3}\cdot 32 = \frac{112}{3} \\ \sum_{s'} P(s'|2,\downarrow)V^\pi(s') &= \frac{2}{3}\cdot 48 + \frac{1}{3}\cdot 24 + 0\cdot 32 = 40 \\ \sum_{s'} P(s'|3,\downarrow)V^\pi(s') &= 0\cdot 48 + \frac{2}{3}\cdot 24 + \frac{1}{3}\cdot 32 = \frac{80}{3} \\ \sum_{s'} P(s'|1,\uparrow)V^\pi(s') &= \frac{1}{3}\cdot 48 + \frac{2}{3}\cdot 24 + 0\cdot 32 = 32 \\ \sum_{s'} P(s'|2,\uparrow)V^\pi(s') &= 0\cdot 48 + \frac{1}{3}\cdot 24 + \frac{2}{3}\cdot 32 = \frac{88}{3} \\ \sum_{s'} P(s'|3,\uparrow)V^\pi(s') &= \frac{2}{3}\cdot 48 + 0\cdot 24 + \frac{1}{3}\cdot 32 = \frac{128}{3}\end{aligned}$$

Then I can derive $\pi'(s)$ as follows.

$$\begin{aligned}\pi'(1) &= \arg\max_a \sum_{s'} P(s'|1,a)V^\pi(s') = \downarrow \\ \pi'(2) &= \arg\max_a \sum_{s'} P(s'|2,a)V^\pi(s') = \downarrow \\ \pi'(3) &= \arg\max_a \sum_{s'} P(s'|3,a)V^\pi(s') = \uparrow\end{aligned}$$

□

8.2 Value and policy iteration

(a) Compute the optimal state value function $V^*(s)$ using the method of value iteration. Print out a list of the non-zero values of $V^*(s)$. Compare your answer to the numbered maze shown below. The correct value function will have positive values at all the numbered squares and negative values at the all squares with dragons.

Sol. I implement the value iteration in *C++* as Code 1. For the initial value of $V_0(s)$, I use the random generator provided by *Eigen* library. Later I update V^* iteratively until the $|V_{k+1} - V_k| < \epsilon$.

The non-zero value are listed in Table 1 with corresponding index, while the figure is drawn with Code 1 and shown as Fig. 1.

3	100.700981
11	102.375264
12	101.523645
15	109.489934
16	110.409033
17	111.335847
20	103.234623
22	106.778267
23	107.674626
24	108.578487
26	112.270440
29	104.101212
30	104.975075
31	105.888536
34	114.163229
35	113.212879
39	103.781407
43	115.121557
47	-133.333333
48	90.985379
49	-133.333333
51	-133.333333
52	116.087929
53	122.024912
56	81.399493
57	93.671656
58	95.172857
59	108.342619
60	109.583651
61	123.643070
62	123.182239
65	-133.333333
66	81.399493
67	-133.333333
69	-133.333333
70	125.249789
71	124.207386
79	133.333333

Table 1: Value of $V^*(s)$

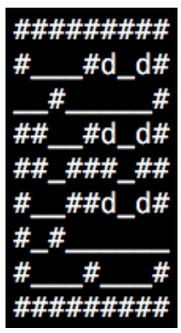

Figure 1: Path Figure Generated by Value Iteration

□

(b) Compute the optimal policy $\pi^*(s)$ from your answer in part (a). Interpret the four actions in this MDP as (probable) moves to the WEST, NORTH, EAST, and SOUTH. Fill in the correspondingly numbered squares of the maze with arrows that point in the directions prescribed by the optimal policy. Turn in a copy of your solution for the optimal policy, as visualized in this way.

Sol. Following (a), I derive the optimal policy for each state as $\pi^*(s)$, and draw the arrow figure as Fig. 2 with Code 1.

□

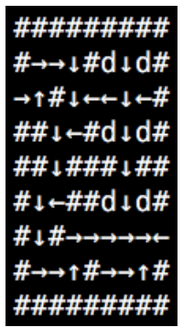

Figure 2: Plan Figure Generated by Value Iteration

(c) Compute the optimal policy $\pi^*(s)$ using the method of policy iteration. For the numbered squares in the maze, does it agree with your result from part (b)? (It should.) Use this check to make sure that your answers in part (a) are correct to at least two decimal places.

Sol. Since I choose a really small epsilon value $\epsilon = 10^{-9}$, (a) and (c) generate exactly the same results for V^* for more than six decimal places. (I omit the table here to save place. Please refer to my appended code, which is shown as Code 1, for more details). □

Also answer the following questions:

(i) if you start with the initial policy that points EAST in every state, how many iterations are required for convergence?

Sol. According to my code, it converges in 4 steps. □

(ii) if you start with the initial policy that points SOUTH in every state, how many iterations are required for convergence?

Sol. According to my code, it converges in 10 steps. □

(d) Turn in your source code for all the above questions. As usual, you may program in the language of your choice.

Sol. I appended my code as Code 1 in Appendix. □

8.3 Effective horizon time

Consider a Markov decision process (MDP) whose rewards $r_t \in [0, 1]$ are bounded between zero and one. Let $h = (1 - \gamma) - 1$ define an effective horizon time in terms of the discount factor $0 \leq \gamma < 1$. Consider the approximation to the (infinite horizon) discounted return,

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \gamma^4 r_4 + \cdots,$$

obtained by neglecting rewards from some time t and beyond. Recalling that $\log \gamma \leq \gamma - 1$, show that the error from such an approximation decays exponentially as:

$$\sum_{n \geq t} \gamma^n r_n \leq h e^{\frac{-t}{h}}.$$

Thus, we can view MDPs with discounted returns as similar to MDPs with finite horizons, where the finite horizon $h = (1 - \gamma) - 1$ grows as $\gamma \to 1$. This is a useful intuition for proving the convergence of many algorithms in reinforcement learning.

Sol. Since $r_t \leq 1$, I can derive that

$$\begin{aligned}\sum_{n \geq t} \gamma^n r_n &= \gamma^t r_t + \gamma^{t+1} r_{t+1} + \cdots \\ &\leq \gamma^t + \gamma^{t+1} + \cdots \\ &= \frac{\gamma^t}{1 - \gamma}.\end{aligned}$$

Here I apply the fact that $\log \gamma \leq \gamma - 1$, then I have

$$\begin{aligned}\log \gamma &\leq \gamma - 1 \\ t \log \gamma &\leq -t \cdot (1 - \gamma) \\ \gamma^t &\leq e^{-t \cdot (1-\gamma)}.\end{aligned}$$

Combine above-mentioned two derivations, I can write

$$\begin{aligned}\sum_{n\geq t}\gamma^n r_n &\leq \frac{\gamma^t}{1-\gamma}\\ &\leq \frac{e^{-t\cdot(1-\gamma)}}{1-\gamma}\\ &= he^{\frac{-t}{h}}.\end{aligned}$$

□

8.4 Convergence of iterative policy evaluation

Consider an MDP with transition matrices $P(s'|s,a)$ and reward function $R(s)$. In class, we showed that the state value function $V^\pi(s)$ for a fixed policy π satisfies the Bellman equation:

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

For $\gamma < 1$, one method to solve this set of linear equations is by iteration. Initialize the state value function by $V_0(s) = 0$ for all states s. The update rule at the kth iteration is given by:

$$V_{k+1}(s') \leftarrow R(s) + \gamma\sum_{s'} P(s'|s,\pi(s))V_k(s').$$

Use a contraction mapping to derive an upper bound on the error

$$\Delta_k = \max_s |V_k(s) - V^\pi(s)|$$

after k iterations of the update rule. Your result should show that the error Δ_k decays exponentially fast in the number of iterations, k, and hence that $\lim_{k\to\infty} V_k(s) = V^\pi(s)$ for all states s.

Sol. Directly calculate the value $\Delta_k = \max_s |V_k(s) - V^\pi(s)|$ can be listed as follows.

$$\begin{aligned}\Delta_k &= \max_s |V_k(s) - V^\pi(s)|\\ &= \gamma\cdot\max_s \left|\sum_{s'} P(s'|s,\pi(s))\cdot V_{k-1}(s') - \sum_{s'} P(s'|s,\pi(s))\cdot V^\pi(s')\right|\\ &= \gamma\cdot\max_s \left|\sum_{s'} P(s'|s,\pi(s))\cdot [V_{k-1}(s') - V^\pi(s')]\right|\\ &\leq \gamma\cdot\max_s \left|\sum_{s'} P(s'|s,\pi(s))\cdot [\max_{s'} V_{k-1}(s') - V^\pi(s')]\right|\\ &= \gamma\cdot\max_s \left|\sum_{s'} P(s'|s,\pi(s))\cdot \Delta_{k-1}\right|\\ &= \gamma\cdot\Delta_{k-1}\\ &= \gamma^k\cdot\Delta_0\end{aligned}$$

Since $\gamma < 1$, it is clear that $\lim_{k\to\infty}\Delta_k \to 0$. In other words, I have

$$\begin{aligned}\max_s |V_k(s) - V^\pi(s)| &\to 0\\ |V_k(s) - V^\pi(s)| &\to 0,\ \forall s.\end{aligned}$$

□

8.5 Value function for a random walk

(a) Assume that the value function $V^\pi(s)$ satisfies a Bellman equation analogous to the one in MDPs with finite state spaces. Write down the Bellman equation satisfied by $V^\pi(s)$.

Sol. With the transition probability table and $R(s) = s$, I have

$$V^\pi(s) = s + \gamma\cdot\left[\frac{2}{5}V^\pi(s) + \frac{3}{5}V^\pi(s+1)\right].$$

□

(b) Show that one possible solution to the Bellman equation in part (a) is given by the linear form $V^\pi(s) = as + b$, where a and b are coefficients that you should express in terms of the discount factor γ. (Hint: substitute this solution into both sides of the Bellman equation, and solve for a and b by requiring that both sides are equal for all values of s.)

Sol. After substitute $V^\pi(s) = as + b$, I can derive

$$as + b = s + \gamma \cdot \left[\frac{2}{5} \cdot (as + b) + \frac{3}{5} \cdot (a \cdot (s + 1) + b)\right]$$
$$as + b = s + \gamma \cdot \left[as + b + \frac{3}{5}a\right]$$

After some simplification, I have

$$as + b - s - as\gamma - b\gamma - \frac{3}{5}a\gamma = 0$$
$$s \cdot (a \cdot (1 - \gamma) - 1) + b \cdot (1 - \gamma) - \frac{3}{5}a\gamma = 0$$

Since for any s, the above formula must hold, I know that $a \cdot (1 - \gamma) = 1$ and thus $a = \frac{1}{(1-\gamma)}$. Later, I can derive $b \cdot (1 - \gamma) = \frac{3}{5} \cdot \frac{\gamma}{1-\gamma}$ and thus $b = \frac{3}{5} \cdot \frac{\gamma}{(1-\gamma)^2}$. □

8.6 Stochastic approximation

(a) Show that the step sizes $\alpha_k = \frac{1}{k}$ obey the conditions

(i) $\sum_{k=1}^{\infty} \alpha_k = \infty$

Sol. It is known as *Harmonic series*, and I can prove it diverges as follows.

$$\begin{aligned}\sum_{k=1}^{\infty} \alpha_k &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \cdots \\ &> 1 + \left(\frac{1}{2}\right) + \left(\frac{1}{4} + \frac{1}{4}\right) + \left(\frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8}\right) + \left(\frac{1}{16} + \cdots \right. \\ &= 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \cdots \\ &= \infty\end{aligned}$$

Thus, $\sum_{k=1}^{\infty} \alpha_k = \infty$. □

and (ii) $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$,

Sol. It is known as *Riemann zeta function*: $\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s}$ when $s = 2$. This series actually can be proved equal to $\frac{\pi^2}{6}$ via *Fourier series*. However, a looser boundary proof can be derived as follows.

$$\begin{aligned}\sum_{k=1}^{\infty} \frac{1}{k^2} &< 1 + \sum_{k=2}^{\infty} \frac{1}{k(k-1)} \\ &= 1 + \sum_{k=2}^{\infty} \left(\frac{1}{k-1} - \frac{1}{k}\right) \\ &= 1 + 1 - \frac{1}{\infty} \\ &= 2 \\ &< \infty\end{aligned}$$

□

thus guaranteeing that the update rule converges to the true mean. (You are not required to prove convergence.)

(b) Show that for this particular choice of step sizes, the incremental update rule yields the same result as the sample average: $\mu_k = \frac{1}{k}(x_1 + x_2 + \cdots + x_k)$. Hint: use induction.

Sol. For $k = 1$, the equality is obvious that

$$\mu_1 = \mu_0 + \frac{1}{1} \cdot (x_1 - \mu_0)$$
$$\mu_1 = \frac{1}{1} \cdot (x_1).$$

Assume that it fulfills $k - 1$ and $\mu_{k-1} = \frac{1}{k-1}(x_1 + \cdots x_{k-1})$, I then have

$$\begin{aligned}\mu_k &= \mu_{k-1} + \alpha_k \cdot (x_k - \mu_{k-1}) \\ &= \frac{k-1}{k} \cdot \frac{1}{k-1} \cdot (x_1 + \cdots + x_{k-1}) + \frac{1}{k} x_k \\ &= \frac{1}{k} \cdot (x_1 + ... + x_k).\end{aligned}$$

□

Appendix

```cpp
#include <cstdio>
#include <cstdlib>
#include <cfloat>
#include <vector>
#include <random>
#include <Eigen/Dense>

using namespace std;
using namespace Eigen;

const int NUM_STAT = 81;
const int NUM_ACTN = 4;

const double g = 0.9925;
const double ceps = 1e-9; // eps for convergence
const double seps = 1e-6; // eps for showing up

default_random_engine generator;
uniform_int_distribution<int> distribution(0,3);

void printPath(const VectorXd& policy);
void printPlan(const VectorXi& pi, const VectorXd& policy);
int valueIterate(const vector<MatrixXd>& prob,
  const VectorXd& reward, VectorXi& pi, VectorXd& policy);
int policyIterate(const vector<MatrixXd>& prob,
  const VectorXd& reward, VectorXi& pi, VectorXd& policy);

int main() {
    // read prob
    vector<MatrixXd> prob(4, MatrixXd::Zero(NUM_STAT, NUM_STAT));
    {
        for (int i = 0; i < 4; i++) {
            string fn = "../dat/prob_a" + to_string(i+1) + ".txt";
            FILE *pf = fopen(fn.c_str(), "r");
            if (pf == NULL) {
                fprintf(stderr, "Cannot open %s\n", fn.c_str());
                exit(EXIT_FAILURE);
            }

            int a, b;
            double v;
            while (fscanf(pf, "%d%d%lf", &a, &b, &v) != EOF) {
                a -= 1;
                b -= 1;
                prob[i](a, b) = v;
            }

            fclose(pf);
        }
    }

    // read reward
    VectorXd reward = VectorXd::Zero(NUM_STAT);
    {
        string fn = "../dat/rewards.txt";
        FILE *pf = fopen(fn.c_str(), "r");
        if (pf == NULL) {
            fprintf(stderr, "Cannot open %s\n", fn.c_str());
            exit(EXIT_FAILURE);
        }

        int d;
        for (int i = 0; i < NUM_STAT; i++) {
            fscanf(pf, "%d", &d);
            reward[i] = d;
        }

        fclose(pf);
    }


    { // 8-2 (a) and (b)
        // random policy
        VectorXd policy = VectorXd::Random(NUM_STAT);
        VectorXi pi = VectorXi::Zero(NUM_STAT);

        printf("\n\nProblem 8-2 (a) and (b)\n");
        int cnt = valueIterate(prob, reward, pi, policy);
        printf("Converge after %d iterations\n", cnt);

        for (int i = 0; i < NUM_STAT; i++) {
            if (policy(i) > ceps or policy(i) < -ceps) {
                printf("#%d: %f\n", i+1, policy(i));
            }
        }

        // 8-2 (a)
        printPath(policy);
```

```
89
90          // 8-2 (b)
91          printPlan(pi, policy);
92      }
93
94
95      { // 8-2 (c)
96          VectorXd policy = VectorXd::Zero(NUM_STAT);
97
98          { // random pi
99              VectorXi pi = VectorXi::Zero(NUM_STAT);
100             for (int i = 0; i < NUM_STAT; i++) {
101                 pi(i) = distribution(generator);
102             }
103
104             printf("\n\nProblem 8-2 (c)\n");
105             int cnt = policyIterate(prob, reward, pi, policy);
106             printf("Converge after %d iterations\n", cnt);
107
108             for (int i = 0; i < NUM_STAT; i++) {
109                 if (policy(i) > ceps or policy(i) < -ceps) {
110                     printf("#%d: %f\n", i+1, policy(i));
111                 }
112             }
113
114             printPath(policy);
115             printPlan(pi, policy);
116         }
117
118         { // 8-2 (c) i.
119             VectorXi pi = VectorXi::Zero(NUM_STAT);
120             for (int i = 0; i < NUM_STAT; i++) {
121                 pi(i) = 2;
122             }
123
124             printf("\n\nProblem 8-2 (c) i.\n");
125             int cnt = policyIterate(prob, reward, pi, policy);
126             printf("Converge after %d iterations\n", cnt);
127             printPath(policy);
128             printPlan(pi, policy);
129         }
130
131         { // 8-2 (c) ii.
132             VectorXi pi = VectorXi::Zero(NUM_STAT);
133             for (int i = 0; i < NUM_STAT; i++) {
134                 pi(i) = 3;
135             }
136
137             printf("\n\nProblem 8-2 (c) ii.\n");
138             int cnt = policyIterate(prob, reward, pi, policy);
139             printf("Converge after %d iterations\n", cnt);
140             printPath(policy);
141             printPlan(pi, policy);
142         }
143     }
144 }
145
146 void printPath(const VectorXd& policy) {
147     printf("\n");
148     for (int i = 0; i < 9; i++) {
149         for (int j = 0; j < 9; j++) {
150             int idx = i + j*9;
151             if (policy(idx) > seps) {
152                 printf("_");
153             } else if (policy(idx) < -seps) {
154                 printf("d");
155             } else {
156                 printf("#");
157             }
158         }
159         printf("\n");
160     }
161 }
162
163 void printPlan(const VectorXi& pi, const VectorXd& policy) {
164     printf("\n");
165     for (int i = 0; i < 9; i++) {
166         for (int j = 0; j < 9; j++) {
167             int idx = i + j*9;
168             if (policy(idx) > seps) {
169                 if (pi(idx) == 0) {
170                     printf("←");
171                 } else if (pi(idx) == 1) {
172                     printf("↑");
173                 } else if (pi(idx) == 2) {
174                     printf("→");
175                 } else {
176                     printf("↓");
177                 }
178             } else if (policy(idx) < -seps) {
179                 printf("d");
```

```
            } else {
                printf("#");
            }
        }
        printf("\n");
    }
}

int valueIterate(const vector<MatrixXd>& prob,
  const VectorXd& reward, VectorXi& pi, VectorXd& policy) {
    bool cont;
    int times = 0;
    do {
        cont = false;
        times += 1;

        vector<VectorXd> pv(4, VectorXd::Zero(NUM_STAT));
        for (int j = 0; j < NUM_ACTN; j++) {
            pv[j] = prob[j] * policy;
        }

        VectorXd npolicy = VectorXd::Zero(NUM_STAT);
        double sum = 0;
        for (int i = 0; i < NUM_STAT; i++) {
            double mmax = -DBL_MAX;
            for (int j = 0; j < NUM_ACTN; j++) {
                if (pv[j](i) > mmax) {
                    mmax = pv[j](i);
                    pi(i) = j;
                }
            }
            npolicy(i) = reward[i] + g * pv[pi(i)](i);

            double diff = fabs(policy(i) - npolicy(i));
            sum += diff;
            if (diff >= ceps or diff <= -ceps) {
                cont = true;
            }
        }
        policy = npolicy;
    } while (cont);

    return times;
}

int policyIterate(const vector<MatrixXd>& prob,
  const VectorXd& reward, VectorXi& pi, VectorXd& policy) {
    bool cont;
    int times = 0;
    do {
        cont = false;
        times += 1;

        MatrixXd id = MatrixXd::Identity(NUM_STAT, NUM_STAT);
        MatrixXd nprob = MatrixXd::Zero(NUM_STAT, NUM_STAT);
        for (int i = 0; i < NUM_STAT; i++) {
            nprob.row(i) = prob[pi(i)].row(i);
        }

        policy = (id - g * nprob).inverse() * reward;

        vector<VectorXd> pv(4, VectorXd::Zero(NUM_STAT));
        for (int j = 0; j < NUM_ACTN; j++) {
            pv[j] = prob[j] * policy;
        }

        int sum = 0;
        for (int i = 0; i < NUM_STAT; i++) {
            double mmax = -DBL_MAX;
            int ppi = pi(i);
            for (int j = 0; j < NUM_ACTN; j++) {
                if (pv[j](i) > mmax) {
                    mmax = pv[j](i);
                    pi(i) = j;
                }
            }

            sum += (ppi != pi(i));
        }
        cont = (sum != 0);
    } while (cont);

    return times-1; // last iteration is the same as previous one
}
```

Listing 1: Code for 8-2