****

**人工智能上机实验报告**

题目：V值预测问题+最优策略

**学生姓名 姜铁**

**学生学号 3015218088**

**学院名称 软件学院**

**专 业 软件工程**

**时 间 2017.10.29**

目 录

第一章、问题描述…………………………………………………×

第二章、程序设计…………………………………………………×

第三章、调试分析…………………………………………………×

第四章、测试分析…………………………………………………×

第五章、未来展望与思考…………………………………………×

### 实验报告编写提纲

1. 问题描述

* 原题表述
* 策略评估：
* 从左上角出发，每个位置都有向下和向右两种action，且若两种action都不出界，则各有0.5概率，否则概率为1.到达右下角会终止并且获得1的收益。此外，每走一步都有+1的收益。
* 找最优解：
* 从左上角出发，在不出界也不触碰障碍的前提下，每个状态有向右和向下两种状态，走到右下角终止，且每一步的reward都为-1.
* 对问题的分析，解决方案的各个环节的功能要求，描述解决方案时，应当注意的条件及限制规则。
* 策略评估：这个问题，每个位置对应一个状态，action即为向下向右两种，每一步的reward为1，未来的折扣为1，在拥有较多的样本后，根据大数定律，结果就会收敛于最终每个状态的value function。

使用mc的方法就是对于能终止的序列统计下每一步开始获得的实际的累积收益，然后用累积收益更新每个状态的value function，并同时统计每个状态的出现次数。

而td的方法试用范围更广，并不要求序列终止，它对于每一个状态都用下一个状态的value值来进行更新，虽然并不全是实际的收益，但是当样本足够多的时候，整个value function会自底（接近终止态的状态）向上（远离终止态的状态）地趋近于真实的value function。

* 找最优解：

我实现的都是on-policy的，首先生成一个随机策略，然后，我们考虑的就是在类比以上的策略评估的基础之上，每次使用episilon-greedy的方法来改进策略。然后由于本次的问题是模型无关的，而原先的基于v(s)的episilon-greedy需要mdp模型，所以我们改为对q(s,a)进行episilon-greedy和评估。实际操作时就是在每次评估后在进行一个episilon-greedy的改进，从而不断逼近最优解。

1. 程序设计

* 主要程序代码及分析（注释，此点很重要）
* 策略评估：

#include <bits/stdc++.h>

using namespace std;

///假定策略为：右下两方向以等概率走一步，若某方向会出界，则必定走另一方向

///走到右下角为止，假定每一步的reward都为1

const int maxn=4;

const double gamma=1.0;

double s[maxn][maxn];///每个状态的总value

int n[maxn][maxn]; ///每个状态出现次数

const int mx[]={1,0};

const int my[]={0,1};

struct Node{int x,y;};

vector<Node> getEpisodes(){///获得一个从起点（左上角）到终点（右下角）的序列

Node start;

start.x=start.y=0;

vector<Node> ret;

ret.push\_back(start);

Node nex=start;

while(!(nex.x==3&&nex.y==3)){

if(nex.y==3)nex.x++;///无法向右走，就必须向下

else if(nex.x==3)nex.y++;///无法向下走，就必须向右

else{///两边都可以走，就随机走一步

int k=rand()%2;

nex.x+=mx[k],nex.y+=my[k];

}

ret.push\_back(nex);

}

return ret;

}

void mc(){

///初始化

memset(s,0,sizeof(s));memset(n,0,sizeof(n));

srand((unsigned)time(NULL));

int sumn=0;

do{

vector<Node> e=getEpisodes();

double G=0;

for(int i=0;i<e.size();i++)G\*=gamma,G+=1;

for(int i=0;i<e.size();i++){

Node ei=e[i];

s[ei.x][ei.y]+=G;

n[ei.x][ei.y]++;

G-=1;

G/=gamma;

sumn++;

}

if(sumn%5==0){///输出中间部分结果

printf("value: \n");

for(int i=0;i<maxn;i++){

for(int j=0;j<maxn;j++){

if(n[i][j]==0)printf("oo ");

else printf("%.4f ",s[i][j]/n[i][j]);

}puts("\n");

}

}

}while(sumn<1e2);

for(int i=0;i<maxn;i++){

for(int j=0;j<maxn;j++){

if(n[i][j]==0)printf("oo ");///未访问过，输出无穷

else printf("%.4f ",s[i][j]/n[i][j]);

}

puts("\n");

}

}

//const double alpha=0.001;

const double alpha=0.1;

double v[maxn][maxn];

void td(){

memset(v,0,sizeof(v));

int sumn=0;

do{

vector<Node> e=getEpisodes();

for(int i=0;i<e.size();i++){

sumn++;

Node ei=e[i];

double r=1;double nexv=0;

if(i+1<(int)e.size()){///不是最后一步，就用下一步的值赋给nexv

Node nexei=e[i+1];

nexv=v[nexei.x][nexei.y];

}

v[ei.x][ei.y]+=alpha\*(r+gamma\*nexv-v[ei.x][ei.y]);

}

if(sumn%100==0){///输出中间部分结果

printf("value :\n");

for(int i=0;i<maxn;i++){

for(int j=0;j<maxn;j++)printf("%.4f ",v[i][j]);

puts("\n");

}

}

}while(sumn<1e4);

puts("\n");

for(int i=0;i<maxn;i++){

for(int j=0;j<maxn;j++){

printf("%.4f ",v[i][j]);

}puts("\n");

}

}

int main(){

//mc();

td();

return 0;

}

* 找最优解：

#include <bits/stdc++.h>

using namespace std;

///假定策略为：右下两方向以等概率走一步

///从左上角开始，走到右下角为止，假定每一步的reward都为-1

const int maxn=4;

const int maxaction=2;

const double gamma=1.0;

double q[maxn][maxn][maxaction];///每个状态的总value

int n[maxn][maxn][maxaction]; ///每个状态出现次数

double f[maxn][maxn][maxaction];///每个状态下向右下两个方向的概率（策略）

const int mx[]={1,0};

const int my[]={0,1};

struct Node{int x,y;};

bool obstcale[maxn][maxn];///纪录某位置是否有障碍

bool inGrid(Node tn){return tn.x>=0&&tn.y>=0&&tn.x<maxn&&tn.y<maxn&&!obstcale[tn.x][tn.y];}///当前位置是否合法

struct QNode{Node qn;int k;};

int getK(Node tn){///取得某个位置的策略，因为f[i][j][0]+f[i][j][1]=1,所以用随机数(0,1)比f[i][j][0]大就用action1，否则用action0

double drand=(double)rand()/RAND\_MAX;

if(drand>=f[tn.x][tn.y][0])return 1;

return 0;

}

vector<QNode> getEpisodes(){///获得一个从起点（左上角）到终点（右下角）的序列

Node start;

start.x=start.y=0;

vector<QNode> ret;

// ret.push\_back(start);

Node nex=start;

while(!(nex.x==3&&nex.y==3)){

int k=getK(nex);

ret.push\_back((QNode){nex,k});

Node tnex=nex;

nex.x+=mx[k],nex.y+=my[k];

if(!inGrid(nex)){

nex=ret[ret.size()-2].qn;

}

}

return ret;

}

void output\_policy(){///输出最终策略

char outs[maxn][maxn];

for(int i=0;i<maxn;i++)for(int j=0;j<maxn;j++)

if(obstcale[i][j])outs[i][j]='#';else outs[i][j]='.';

///获取并记录每个状态的最优action

int best[maxn][maxn];

for(int i=0;i<maxn;i++){

for(int j=0;j<maxn;j++){

int maxk=0;double maxp=f[i][j][maxk];

for(int k=0;k<maxaction;k++)if(f[i][j][k]>maxp)

maxp=f[i][j][k],maxk=k;

best[i][j]=maxk;

}

}

///保存最优路线上经过的点

bool needout[maxn][maxn];

memset(needout,0,sizeof(needout));

Node nex;nex.x=nex.y=0;

while(!(nex.x==3&&nex.y==3)){

needout[nex.x][nex.y]=true;

outs[nex.x][nex.y]= best[nex.x][nex.y]==0? 'd':'r';

int k=best[nex.x][nex.y];

nex.x+=mx[k],nex.y+=my[k];

}

outs[maxn-1][maxn-1]='e';

for(int i=0;i<maxn;i++){

for(int j=0;j<maxn;j++){

if(needout[i][j]){///若是最优路线上的点

int maxk=best[i][j];

if(maxk==0)printf(" ↓ ");else printf("→ ");

}

else{

printf(" %c ",outs[i][j]);

}

}puts("\n");

}

}

void mc(){

///初始化

memset(q,0,sizeof(q));memset(n,0,sizeof(n));

///初始化每个状态下的走每个action的概率都相同

for(int i=0;i<maxn;i++)for(int j=0;j<maxn;j++)for(int k=0;k<maxaction;k++)

f[i][j][k]=1/(double)maxaction;

memset(obstcale,0,sizeof(obstcale));

///设置障碍

obstcale[3][1]=true;obstcale[3][2]=true;

int cnt=0;

do{

vector<QNode> e=getEpisodes();

double epsilon=1/(double)(++cnt);

double G=0;

for(int i=0;i<e.size();i++)G\*=gamma,G+=-1;

for(int i=0;i<e.size();i++){

///update n&&q

int x=e[i].qn.x,y=e[i].qn.y,k=e[i].k;

q[x][y][k]+=G;

n[x][y][k]++;

G-=-1;

G/=gamma;

///update policy

int maxk=0;double maxp=f[x][y][maxk];

for(int tk=0;tk<maxaction;tk++)if(f[x][y][tk]>maxp)

maxk=tk,maxp=f[x][y][tk];

if(maxk!=k&&

q[x][y][k]\*n[x][y][maxk]>q[x][y][maxk]\*n[x][y][k]){

for(int tk=0;tk<maxaction;tk++)

if(tk==k) f[x][y][tk]=1-epsilon+epsilon/maxaction;

else f[x][y][tk]=epsilon/maxaction;

}

}

}

}while(cnt<1e4);

///print the policy

output\_policy();

}

const double alpha=0.1;

double v[maxn][maxn][maxaction];///保存sarsa中每个q(s,a)

void sa(){

///初始化

memset(v,0,sizeof(v));

for(int i=0;i<maxn;i++)for(int j=0;j<maxn;j++)for(int k=0;k<maxaction;k++)

f[i][j][k]=1/(double)maxaction;

memset(obstcale,0,sizeof(obstcale));

///设置障碍

obstcale[3][1]=true;obstcale[3][2]=true;

int cnt=0;

do{

vector<QNode> e=getEpisodes();

double epsilon=1/(double)(++cnt);

for(int i=0;i<e.size();i++){

QNode ei=e[i];

int x=ei.qn.x,y=ei.qn.y,k=ei.k;

double r=-1;double nexv=0;

if(i+1<(int)e.size()){

QNode nexei=e[i+1];

nexv=v[nexei.qn.x][nexei.qn.y][nexei.k];

}

v[x][y][k]+=alpha\*(r+gamma\*nexv-v[x][y][k]);

///update policy

int maxk=0;double maxp=f[x][y][maxk];

for(int tk=0;tk<maxaction;tk++)if(f[x][y][tk]>maxp)

maxk=tk,maxp=f[x][y][tk];

if(maxk!=k&&

v[x][y][k]>v[x][y][maxk]){

for(int tk=0;tk<maxaction;tk++)

if(tk==k) f[x][y][tk]=1-epsilon+epsilon/maxaction;

else f[x][y][tk]=epsilon/maxaction;

}

}

}while(cnt<1e4);

///print the policy

output\_policy();

}

int main(){

srand((unsigned)time(NULL));

sa();

// mc();

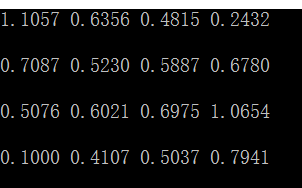
return 0;

}

1. 调试分析（可选）

问题：使用蒙特塔罗做评估时程序异常终止退出

原因：有状态没有出现过,v[i]=s[i]/n[i]而n[i]=0导致的。

解决：特判了n[i]=0的情况并对于n[i]=0时输出无穷。

问题：如右图，输出一段不理想且显然不正确的结果

原因：循环次数太少，还没有收敛

解决：将循环次数翻了几倍就很接近正确答案了。

问题：程序一直不输出结果，最后还显示bad\_alloc…..

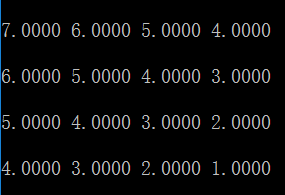
原因：getEpisodes中会出现撞到obstcle或者出界的情况而导致死循环

解决：特判下然后做后退处理

问题：程序运行结果跟初始化一致。。。中间的过程貌似没有产生什么影响

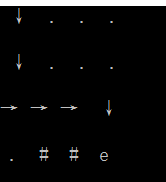
原因：由于sa中很多部分是复制mc中的，忘记改数组名了，结果还是对原来mc中的数组做了修改。。。

解决：把数组名称更改即可

1. 结果分析

策略评估：

如右图，从左上角到右下角，每一步有1的收益，到达右下角又有1的额外收益，所以右下角的value为1，而每个位置只有向下向右两个动作，所以结果如图所示。

找最优解：

如右图，’#’表示障碍，从左上角的最优路线为向下，向下，向右，向右，向右，向下。

五、未来展望与思考

* 这次的实验使我对mc和sarsa的on-policy的最优解寻找以及策略评估有了深刻的认识，也让我对自己的学习方法有了反思。Mc对于我还是比较好理解的，但是sarsa的算法就一直都没能看懂。看不懂的主要原因在于sarsa中的episilon-greedy的方法一时没能记得，又从前面再学了一遍才了解。
* 本次实验，我发现了强化学习真的是一个很神奇的东西，在对一个问题一无所知的情况下通过探索，竟然能够像人类一样不断地利用外界给予的反馈增长经验，学习、了解甚至以最优的策略来应对外界环境。但是，有一个问题在于，我们人在处理问题时很多重大的问题都是之前没有什么探索经验的，是独一无二的，或许计算机将来能够发展出一种类比迁移的算法，能够根据问题的相似度匹配出最接近的历史发生过的问题并根据那件发生过的问题的最优解类比地找出当前问题的最优解也说不定呢。