强化学习

张运吉 (211300063、211300063@nju.smail.edu.cn)

(南京大学人工智能学院, 南京 210093)

1 阅读代码，阐述强化学习的方法和过程。并且回答以下问题：

强化学习，是在与环境的互动中为了达成一个目标而进行的学习过程。强化学习的组成元素有:Agent(与环境交互的主体)，Environment（智能体以外的一切，主要由状态集合组成），State（一个表示环境的数据，状态集则是环境中所有可能的状态），Action（智能体可以做出的动作，动作集则是智能体可以做出的所有动作），Reward（智能体在执行一个动作后，获得的正/负反馈信号，奖励集则是智能体可以获得的所有反馈信息），Policy（强化学习是从环境状态到动作的映射学习，称该映射关系为策略。通俗的理解，即智能体如何选择动作的思考过程称为策略），Goal（智能体自动寻找在连续时间序列里的最优策略，而最优策略通常指最大化长期累积奖励）。

代码整体理解：act 方法：act 方法主要做两件事情，首先调用 learnPolicy 方法完成强化学习的过程，然后根据强化学习得到的策略返回一个最佳 action。learnPolicy 方法：在这个函数中进行了 10 此迭代，每次迭代都调用 simulate 函数来模拟游戏，模拟最大深度为 20 或者游戏结束，simulate 过程如下：首先根据 epsilon-greedy 策略 silon 概率随机选取一个动作，1-epsilon 概率选取 Q 值最大的动作）选取一个动作，对当前 state 执行这个 action，然后通过公式

$$Q(s,a)+=\alpha[r+\gamma\, max\, Q(s_{new})-Q(s,a)]$$

更新 Q(s,a)的值，把数据记录在 sequence 中。最后框架代码通过 mpolicy 的方法 fitQ 训练策略，fitQ 会把模拟的数据当作输入，调用 weka 内置的 REPTree 分类器进行训练。

1.1 策略模型用什么表示？该表示有何缺点？有何改进方法？

策略模型是利用 epsilon-grededy 的 Q-learning 方法。也就是我们上面介绍的，有 epsilon 概率随机选取动作，有 1-epsilon 概率选择 Q 值最大的动作，并通过上述公式更新 Q 值。然后使用 fitQ 函数来调用 weka 的 REPTree 模型来训练策略。

该表示的缺点是：数据量大情况下更新 Q-Table 的时间和空间开销大，而且随着探索的深入，epsil-greedy 策略不再适用，因为刚开局时探索范围小，不确定性大，随机探索效果更好，而到了游戏后期，探索的概率应该降低，要更注重最优动作。

改进方法:随着探索的深入动态更新 epsilon 的值;也可以使用 Double Q-learning 方法;也可以考虑 SARSA 方法，先执行 action 在返回 Q 值。

1.2 Agent.java 代码中 SIMULATION_DEPTH, m_gamma, m_maxPoolSize 三个变量分别有何作用？

SIMULATION_DEPTH 是模拟的最大深度，m_gamma 是折扣因子，m_gamma 越大说明其越重视未来的回报奖励，m_maxPoolSize 是 m_dataset 的大小，即存储的最大状态数，当 Instance 个数超过 m_maxPoolSize 时，会删去前面的 Instance，保证了数据分布的随机性。

1.3 QPolicy.java 代码中，getAction 和 getActionNoExplore 两个函数有何不同？分别用在何处？

getAction 函数使用了 epsilon greedy 策略，而 getActionNoExplore 没有使用此策略（总是返回 Q 值最大的 Action），下图是 getAction 函数比 getActionNoExplore 函数多的部分：

```
// epsilon greedy
if( m_rnd.nextDouble() < m_epsilon ){
    bestaction = m_rnd.nextInt(m_numActions);
}
```

getAction 函数用在 simulate 函数中进行模拟探索，因为探索时需要一定的随机性，从而可以避免贪心算法的缺陷，提高找到最优解的概率；getActionNoExplore 函数用在 act 函数中，是在已经完成探索的情况下，在决策阶段选择最优的动作。

2 尝试修改特征提取方法，得到更好的学习性能，并报告修改的尝试和得到的结果

框架代码中的原有特征是记录了屏幕上所有的位置信息以及 4 个游戏状态:GameTick, AvatarSpeed, AvatarHealthPoints, AvatarType，还有动作信息和奖励信息。

2.1 一处小修改

框架代码的 getAction 函数中，epsilon greedy 策略被放在了后面，这样的缺点是：如果最后选择的是随机动作，那么前面计算出来的 Q 值最大的动作就没用用处，这就浪费了时间。所以我把 epsilon greedy 策略放到了计算最大 Q 值动作的前面。如下图：

```
double[] Q = getQArray(feature);

// find best action according to Q value
int bestaction = 0;
// epsilon greedy
if( m_rnd.nextDouble() < m_epsilon ){
    bestaction = m_rnd.nextInt(m_numActions);
    return bestaction;
}

for(int action=1; action<m_numActions; action++){
    if( Q[bestaction] < Q[action] ){
        bestaction = action;
    }
}
// among the same best actions, choose a random one
int sameactions =0;
for(int action=bestaction+1; action<m_numActions; action++){
    if(Q[bestaction] == Q[action]){
        sameactions++;
        if( m_rnd.nextDouble() < 1.0/(double)sameactions )
            bestaction = action;
    }
}
return bestaction;
```

2.2 我的修改

首先，我增加了 Avatar 和目标的距离（这里使用曼哈顿距离）。

```
// 这两个变量代表avatar距离目标的x，y距离
double xToTarget = 0;
double yToTarget = 0;
if(obs.getPortalsPositions()!=null)
    for(ArrayList<Observation> l:obs.getPortalsPositions())
        allobj.addAll(l);
for(Observation obj : allobj){
    Vector2d p = obj.position;
    int x = (int)(p.x/28);
    int y= (int)(p.y/28);
    map[x][y] = obj.itype;
    if(obj.itype==4) {
        xToTarget = Math.abs(x-avatar_x);
        yToTarget = Math.abs(y-avatar_y);
    }
}
```

其次，我增加了 Avatar 的前方是否有障碍物这一特征，防止其被卡在障碍物前.

```
double up = 100;
if( obs.getImmovablePositions()!=null ) {
    for (ArrayList<Observation> l : obs.getImmovablePositions()) {
        allobj.addAll(l);
        for(Observation obj:l){
            if(avatarY==obj.position.y+28)
                frontImmoving=Math.min(frontImmoving,Math.abs(obj.position.x-avatar_x));
            if(obj.position.x==avatar_x && obj.position.y+28==avatarY)
                up=0;
        }
    }
```

最后，减少框架代码中的一些无用特征，因为 Avatar 在某一时刻并不需要过于关注“远方”的情况，只需要关注“附近”的情况即可，所以我们可以不用记录整个地图的信息，只需记录 Avatar 前四行和后两行的信息（记录后两行是因为 Avatar 可能要往后撤躲避危险）。

```
        for(int y=avatar_y - 2; y<avatar_y + 4; y++)
            for(int x=0; x<28; x++)
                feature[y*28+x] = map[x][y];
```

同时也要在 dataHeader 中添加相应的 Attritube：

```
Attribute att = new Attribute( attributeName: "GameTick" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarSpeed" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarHealthPoints" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarType" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarToTarget" ); attInfo.addElement(att); // 与目标距离
att = new Attribute( attributeName: "haveObstacles" ); attInfo.addElement(att); // 前方是否有障碍物体
```

2.3 修改效果

不修改前， Avatar 只会待在最下面两层。修改之后， Avatar 学会了躲避移动的物体，并且会向上冲，最远能到达最后一层阻挡，但常常由于血量原因，当吃了一次亏掉下来后它就很保守地移动，而非大胆地向上探索了。

3 尝试修改强化学习参数，得到更好的学习性能，并报告修改的尝试和得到的结果

3.1 评估函数

查看框架代码，使用的是 WinScoreHuristic 评估函数，这个评估函数很简单，就是赢了返回 1000，输了返回-1000，没有使用上其他有用的信息，所以我对其进行了拓展。具体是，增加 Avatar 和目标的距离，Avatar 的血量，距离 Avatar 最近的移动物体和距离不能移动物体。

以下是修改后的部分代码：

```
public double evaluateState(StateObservation stateObs) {
    boolean gameOver = stateObs.isGameOver();
    Types.WINNER win = stateObs.getGameWinner();
    // 在游戏得分的基础上加一点其他考虑因素
    double rawScore = stateObs.getGameScore();

    // 赢了或者输了直接返回，不然返回rawScore
    if(gameOver && win == Types.WINNER.PLAYER_LOSES)
        return HUGE_NEGATIVE;

    if(gameOver && win == Types.WINNER.PLAYER_WINS)
        return HUGE_POSITIVE;

    // return rawScore;
    // 获取rawScore的过程（avatar位置，avatar与目标距离，avatar距离最近的移动和非移动物体，游戏得分，avatar血量）
    Vector2d avatarPos=stateObs.getAvatarPosition();
    ArrayList<Observation>[] movingObj=stateObs.getMovablePositions();
    ArrayList<Observation>[] immovingObj=stateObs.getImmovablePositions();
    ArrayList<Observation>[] protal=stateObs.getPortalsPositions();
```

```
    for(ArrayList<Observation> l:movingObj){
        if(l.size()>0){
            for(Observation obs:l){
                if(movingDist>avatarPos.dist(obs.position)) {
                    minMoving = obs.position;
                    movingDist=avatarPos.dist(obs.position);
                }
            }
        }
    }
    if(protal!=null){
        protalPos=protal[0].get(0).position;
        protalDist=avatarPos.dist(protalPos);
    }
    if(movingDist==0)
        rawScore += stateObs.getAvatarHealthPoints()+stateObs.getGameScore()-protalDist*100-999
                +immovingDist;
    else
        rawScore += stateObs.getAvatarHealthPoints()+stateObs.getGameScore()-protalDist*100
                +movingDist*10+immovingDist;
    return rawScore;
}
```

3.2 SIMULATION_DEPTH

直觉上可能会觉得 SIMULATION_DEPTH 越大越好，但事实上不是的。这是因为我们上面提到的，Avatar 不需要关注“太远”处的情况，只需要关注“附近”的情况，所以我尝试把 SIMULATION_DEPTH 调小，发现调整为 8 左右比较好。

3.3 m_gamma

此参数主要的作用是给 reward 衰减，因为 reward 只是对局面的评估，不一定是实际上获得到奖励。框架代码中这个参数设为 0.99，我觉得太大了，于是我尝试调小，发现 0.8 左右效果较好。

3.4 epsilon

理论上，epsilon 这个值太大或者太小都不好，太大的话产生随即动作的概率变大，不能充分利用已学习的知识，太小又会容易选择已经选取过的动作，框架代码给的初值是 0.3，经过反复测试，我发现 0.3 比较不错，所以我保留了这个值。

4 结束语

在本次实验中，更加深入具体的了解了强化学习的过程与 Q-Learning 算法的整体运行流程。对特征参数的提取和启发式函数的设计有了进一步的了解。同时根据调整参数提高学习性能，结合游戏理解每个参数的含义。