

作业 4: FreeWay 游戏

孙嘉欣 (201240005、201240005@smail.nju.edu.cn)

(南京大学 匡亚明学院大理科班)

摘要: 本次作业目的是使用强化学习来自主玩 FreeWay 游戏.报告中首先阐述了强化学习 Q-learning 模型的方法,以及框架代码中的实现过程,具体回答三个关键变量、两个关键函数的作用.然后尝试通过特征提取方法、Heuristic 函数、强化学习参数,三个部分的修改,来提升强化学习的学习性能.

关键词: 强化学习;Q-learning; Epsilon greedy

1 强化学习的方法和过程描述

1.1 策略模型

强化学习是利用观察到的回报来学习针对某个环境的最优（或接近最优）策略.本次作业中,策略模型用 Q-learning 模型表示.Q-learning 模型使用行动-价值函数 Q 函数,来表示在给定状态下采取特定行动的期望效用.该函数基于 Bellman 方程,公式如图 1^[1].

The diagram shows the formula $Q^\pi(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t]$. Three red arrows point from parts of the formula to labels below:
1. From $Q^\pi(s_t, a_t)$ to "Q-Values for the state given a particular state".
2. From the expectation operator $E[\dots]$ to "Expected discounted cumulative reward".
3. From $| s_t, a_t$ to "Given the state and action".

Fig.1 Formula of Q-value
图 1 Q 函数计算公式

不同状态-行动 Q 值构建成一个 Q 表.初始化每个 Q 值都为 0,当开始探索环境时,基于当前所得到的奖励和对下一步的最大估计,不断更新表中的 Q 值,Q 值的更新公式如图 2^[2].

The diagram shows the update formula $newQ_{S,A} = Q_{S,A} + \alpha (R_{S,A} + \gamma * \max Q'(s', a') - Q_{S,A})$. Brackets and labels identify the components:
- $newQ_{S,A}$: 基于状态和行动的新Q值
- $Q_{S,A}$: c当前Q值
- α : 学习效率
- $R_{S,A}$: 基于状态和行动的奖励
- γ : 折扣因子
- $\max Q'(s', a')$: 在给定的状态和行动下未来最大的奖励

Fig.2 Updating formula of Q-value
图 2 Q 值更新公式

每次采取行动时,根据当前观察模拟后,更新所得的 Q 值来选取能够获得最大收益的动作.

缺点:不关心遵循的实际策略,是一个脱离策略的学习算法.

改进方法:可以使用 SARSA(State-Action-Reward-State-Action)模型改进.SARSA 等到实际采取行动后再回传那个行动的 Q-值,当发生探索时,SARSA 依附策略,更现实.

1.2 实现过程

Agent 决策的具体过程:先迭代模拟一定次数(框架代码为 10 次),每次模拟收集数据,并更新数据集这个过程也就是 Agent 思考的过程.模拟迭代结束后,将脱离实际策略的 Q-learning 策略分类器作用到更新好的数据集上,用于决策当前的实际问题.最终根据所得的 Q 值来选取能够获得最大收益的动作.

思考决策时的模拟由 `simulate` 函数完成,通过循环,在当前局面下向下模拟一定步数.每次都完成如下步骤:通过 `RLDataExtractor` 提取局面特征、通过 `QPolicy` 在平衡经验和探索后做出下一步决策、通过启发式函数评估局面的得分变化、将记录得分变化.最后,通过记录的数据,更新 Q 值.

1.3 变量作用

`SIMULATION_DEPTH` 是思考决策时的模拟深度,作用于 `simulate` 函数中,在当前局面下向下模拟 `SIMULATION_DEPTH` 步,然后计算对应状态的 Q 值与累计奖赏值.

`m_gamma` 是折扣因子 γ ,作用是计算效应值时的折扣累计奖赏. γ 定义了未来奖励的重要性,越接近 1,越重视后续状态的价值,越接近 0,越重视当前利益的影响.

`m_maxPoolSize` 是记忆容量,作用是限制数据集的最大实例数,更新数据集时,超出容量就会删掉最早的记忆.

1.4 函数作用

`QPolicy.java` 代码中,`getAction` 和 `getActionNoExplore` 两个函数的不同在于是否应用 `epsilon greedy` 策略.两个函数都遍历 Q 值,得到其中最佳动作,并考虑了 Q 值相同的动作,通过随机数保证相同 Q 值的动作获取概率相同.`getAction` 比 `getActionNoExplore` 多使用了 `epsilon greedy` 策略,设置了一个执行探索的概率 `m_epsilon` (初始设定为 0.3),使得选择 action 时,存在 `m_epsilon` 的概率不选择得分最好的动作,而是随机选择一个动作,作为探索,来在探索和利用间进行折中.

当 Agent 在脑中模拟时,使用 `getAction` 函数,进行探索,即用在 Agent 的 `simulate` 函数中;当 Agent 实际行动时,就不再随机探索,而是根据思考好的策略,使用 `getActionNoExplore` 函数,选择得分最好的动作.

2 特征提取方法的改进

2.1 现有的特征提取方法

原有的特征提取方法是记录当前游戏画面上每个位置的信息(即每个坐标上的物体种类 `itype`),游戏画面为 `32*28`,共 868 个位置,以及 4 个游戏状态信息(游戏时间 `GameTick`、Avatar 的速度 `AvatarSpeed`、血量 `AvatarHealthPoints` 和状态 `AvatarType`).训练数据集内保存每一时刻提取到的游戏画面特征、玩家操作和局面对应 Q 值.玩家操作有 4 种,通过一个分类变量 `class` 表示,0 到 3 分别对应左、右、下、上.

2.2 修改尝试

2.2.1 首次修改

- 1) 增加 Avatar 与传送门 Portal 的距离:`StateObservation` 的 `getPortalsPositions()` 方法可以获取传送门.这里只有一个传送门,所以位置是 `obs.getPortalsPositions()[0].get(0).position`.但传送门位置会随机变化,有时候传送门会短暂消失.若 Portal 目标不存在,则假设其位置在第一行中间(`15*28.0,28.0`),以此来计算距离,代码见图 3.

```
Vector2d Avatar_p= obs.getAvatarPosition();
System.out.println(Avatar_p);
Vector2d Portal_p=new Vector2d( x: 15*28.0, y: 28.0); // 目标位置, 若目标不存在, 假设在第一行中间
if(obs.getPortalsPositions()!=null)
    Portal_p=obs.getPortalsPositions()[0].get(0).position;

double goalDis=ManDis(Portal_p,Avatar_p); //distance between Avatar and portal
```

Fig.3 Code about goalDis

图 3 目标距离代码

- 2) 增加最近 Track 的距离: 游戏中有 4 种 Track, 都是 Movable objects, 遍历所有获取的 MovablePositions, 即可找到距离最近的 Track. 代码见图 4.

```
double minTrackDis=100; //initial
if( obs.getMovablePositions()!=null ){
    for(ArrayList<Observation> l : obs.getMovablePositions()) {
        allobj.addAll(l);
        for(Observation o:l){
            double MD=ManDis(o.position,Avatar_p); //计算曼哈顿距离
            if(MD<minTrackDis) //找最近的Track
                minTrackDis=MD;
        }
    }
}
```

Fig.4 Code about NearestTrackDis
图 4 最近 Track 距离代码

以上距离都用曼哈顿距离表示.

性能: 还是很难过马路, 会停留在右下角徘徊, 但相较于原先的方法, 会引导 Avatar 在水平距离上向传送门 Portal 靠近. 躲车辆的表现也稍微好了一点.

2.2.2 二次修改

增加最近 Track 的种类: 游戏中有 4 种 Track, 分别是向左、向右的快车和慢车, 车速和行车方向都对躲闪有影响, 因此找到最近 Track 后同时记录 Track 的种类.

性能: 不会主动往车辆驶来的方向撞, 而是会反向躲避. 但左右躲避的增加, 会减少上下躲闪, 不一定有利.

2.2.3 三次修改

将记录最近 Track 的距离, 改为记录向 Avatar 方向行使的最近车辆距离: 光考虑距离, 有的车辆可能已经过去, 不会撞到. 4 种 Track 的 itype 分别是: fastRtruck-7, slowRtruck8, fastLtruck-10, lowLtruck-11. 根据 itype 和横坐标, 判断是否正朝向 Avatar 行使, 代码见图 5.

```
double minTrackDis=100; //initial
int trackType=0;
if( obs.getMovablePositions()!=null ){
    for(ArrayList<Observation> l : obs.getMovablePositions()) {
        allobj.addAll(l);
        for(Observation o:l){
            boolean direct=true;
            if(o.position.x>Avatar_p.x){ //right-->
                if(o.itype==7||o.itype==8) //fastRtruck-7,slowRtruck8
                    direct=false;
            }
            else if(o.position.x<Avatar_p.x){ //left<--
                if(o.itype==10||o.itype==11) //fastLtruck-10,lowLtruck-11
                    direct=false;
            }
            if(direct){
                double MD=ManDis(o.position,Avatar_p); //计算曼哈顿距离
                if(MD<minTrackDis) { //找最近的Track
                    minTrackDis = MD;
                    trackType= o.itype;
                }
            }
        }
    }
}
```

Fig.5 Code about Nearest Track directing Avatar
图 5 向 Avatar 方向行使的最近车辆

性能:前方车辆未靠近时不会急着躲闪,会左右横跳,等到靠近时才躲闪.一开始很快移动一阵,受伤后会变得保守不向,停留在最下方一行.

2.2.4 四次修改:修正 BUG

后期加入调参 Heuristic 函数,性能提高后,发现 Avatar 即使到达最上方也不会向传送门靠近,并且获取 Portal 会抛出异常,显示列表 Index 越界.发现原因是 Avatar 遇到目标后 Portal 物体会被删除,列表变空.因此又修改了特征提取函数,加入对列表大小的判断.

```
Vector2d Avatar_p = obs.getAvatarPosition();
Vector2d Portal_p = new Vector2d( x: 15 * 28.0, y: 28.0); // 目标位置, 若目标不存在, 假设在第一行中间
if (obs.getPortalsPositions() != null) {
    for (ArrayList<Observation> l : obs.getPortalsPositions()) {
        if (l.size() > 0) {
            Portal_p = l.get(0).position;
        } else
            Portal_p = Avatar_p;
    }
}
goalDis=ManDis(Portal_p,Avatar_p); //distance between Avatar and portal
```

Fig.8 Debugged code for goalDis

图 8 修正后的目标距离代码

3 Heuristic 函数的修改

先是尝试在不修改 Heuristic 函数的情况下调参,但模型都无法取胜,且行为变化不规律,不好观察.原先的 Heuristic 函数只于胜负有关,由于模拟深度有限,得分很多都是 0,没有变化,在还没有取胜前都难以判断奖惩.因此修改了 Heuristic 函数.

考虑在未取胜时血量、用时、前进程度(用 Avatar 的纵坐标代表),修改为 $\text{rawScore} + \text{hp} * 100 - \text{time} - \text{AvatarPosition}().y$.修改后性能提升明显,可以成功过几排马路.从上到下,依次编号为 1-4 车道.第一次尝试时可以到达 2 车道,一次受伤后还可以尝试到达 3 车道,再次受伤后就会逐渐停留在最下方,后期学到的信息都会使 Avatar 停留在最下方一排.

后来随着调参,性能进一步提高,发现 Avatar 即使到达最上方也不会向传送门靠近.输出 Heuristic 函数的返回情况,原因是无法此时游戏没有结束,不能通过胜负分判断,必须修改 Heuristic 函数,靠自己构造的函数判断.因此,为了让 Avatar 能到达目标,又一次修改 Heuristic 函数,改为 $\text{rawScore} + \text{hp} * 100 - \text{time} - \text{goalDis}$.将前进程度用 Avatar 与传送门 Portal 的距离表示.

4 强化学习参数的调整

Table 1 Original parameters
表 1 原始参数

变量	初始值
模拟深度 SIMULATION_DEPTH	20
记忆容量 m_maxPoolSize	1000
折扣因子 m_gamma	0.99
探索系数 m_epsilon	0.3

原始参数如表 1 所示.由于学习模型训练太慢,首先需要调整和训练时间有关的参数 SIMULATION_DEPTH 和 m_maxPoolSize,缩短游戏时间,所以无法有很大的模拟深度和记忆容量,无法模拟完一整关游戏,只能搜索在一定车道范围内的情况.不同关的前期过马路情况相似,但前一关的早期数据也保存不到后一关就会被删除,因此能学习到的经验并不足够长期.这样看来,能保存下的长期经验不如短期记忆重要.

从游戏特征上看, Freeway 游戏在一关内要经过 4 个车道, 只有经过前面的车道才能到达后面, 而前后的奖励没有太大关联, 因此可以看作是阶段性的游戏. 由于模拟深度有限, 无法模拟完全局游戏, 只能模拟通过一定车道, 所以在前期, 未来到达在后几个车道获得的奖励, 远没有通过当前车道的奖励重要, 当 Avatar 成功过马路时应该及时鼓励. 所以 m_gamma 应该较小, 折扣累计奖赏时更重视当前利益. 同时, Freeway 变化较大, 传送门的位置会变化, 节奏比较快, 但经过每个车道的策略都是类似的, 因此短期经验就足够, 不需要特别长期的经验, 记忆容量可以减小, 删掉前期的经验. 并且要加大探索概率, 来适应 Freeway 的变化.

4.1 模拟深度 SIMULATION_DEPTH

初始设定为 20 时, 决策时间过长. 修改为 10 后, 不仅决策时间缩短, 而且性能也有提高, 原因可能是模拟次数较少时, 收集到的受伤数据较少, 次数增多后收集到的受伤数据比例增大, 使 Avatar 更保守, 表现也更差.

后面的参数调整时, 都基于 SIMULATION_DEPTH = 10.

4.2 记忆容量 $m_maxPoolSize$

初始设定为 1000 时, 过大的记忆会拖慢训练速度, 经验积累得较多时策略也会比较保守, 一旦受伤停留在右下角后, 就会一直在右下角徘徊. 尝试设定为 800、600、500, 最终设定为 500 时, 训练速度比较快, 且不会一直缩在右下角, 游戏后期也会有一定倾向前进.

4.3 折扣因子 m_gamma 和探索系数 $m_epsilon$

m_gamma 和 $m_epsilon$ 协同作用, 降低未来奖励的同时需要加大探索比例, 否则模型会变得非常短视; 加大探索的同时, 重视短期奖励, 也可以充分发挥探索的作用.

$m_gamma=0.9$ 时, 已经有一定几率在长时间移动试探后到达 2 车道, 接近最上方的 1 车道, 但受伤后还是会停留在右下角. 0.8 时开始有取得胜利的情况. 之后的尝试组合可以用多局得分和用时来表示性能, 每种组合玩 4 场游戏的取最大得分, 得分相同时比较用时, 4 场游戏得分取均值, 如表 2.

Table 2 Trial combinations
表 2 尝试组合

变量	$m_epsilon$					
m_gamma	最大得分(用时) /平均得分	0.3	0.4	0.6	0.8	0.9
0.6		10(284) /1	10(768) /2.5	20(364) /7.5	20(453) /12.5	-
0.4		-	30(323) /10	30(523) /15	50(874) /40	-
0.2		-	-	20(341) /7.5	50(473) /36	80(789) /32.5
0.1		-	-	-	30(335) /22.5	70(398) /47.5

性能较最好的组合是 $m_gamma=0.1, m_epsilon=0.9$. 总体上, 适当地 m_gamma 减小、 $m_epsilon$ 增大会提高得分, 但要同步调整, m_gamma 相对过小或 $m_epsilon$ 相对过大, 得分或稳定性反而会下降.

5 结束语

实验通过特征提取方法、Heuristic 函数、强化学习参数, 三个部分的修改, 来提升强化学习的学习性能. 由于框架代码 Heuristic 函数的缺陷, 比作业原先的要求多修改了这部分. 光依靠修改特征提取方法和 Heuristic 函数获得的优化有限, 但这两部分会是后面调整强化学习参数的基础. 调参对强化学习效果的影响最大, 随着调参, 学习性能提高, 特征提取方法和 Heuristic 函数又会限制学习的效果, 呈现出一系列问题. 此时回头去进一步修改特征提取方法和 Heuristic 函数, 最终能取得学习性能较好的模型.

References:

- [1] <https://easyai.tech/ai-definition/q-learning/>

[2] <https://zhuanlan.zhihu.com/p/35724704>