FreeWay 实验报告

宋磊（181220049、974534426@qq.com）

(南京大学 人工智能学院)

1 强化学习的方法和过程

1.1 强化学习的过程

1.1.1 主要思路

强化学习使用了基于 epsilon greedy 的 Q-learning 算法，同时融入了 DQN 的思想。但由于状态过多，选择最优 action 时，使用函数近似 function approximation（weka 中的 REPTree）的方法得到当前状态对应所有 action 的 Q 值，然后使用 epsilon greedy 的策略选择 action，并且依照下图的方程更新 Q 值，并用新的 Q 值和对应的状态数据对训练 REPTree 的数据进行随机更新（这里使用了经验回放 experience replay），得到新的 REPTree，不断重复，使得 Q 表收敛得到最优解。

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

1.1.2 详细分析

在 act()函数的开始先调用 learnPolicy()进行学习，在 learnPolicy()内进行十次迭代，每一轮使用 simulate()函数进行模拟之后的行动（向前看 SIMULATION_DEPTH 步）。

在 simulate()中，使用 epsilon greedy 的策略选择每一步行动，在当前状态下不断向前探索，并将每一步行动后 Q 值的增量与衰减因子γ^t的乘积及相应的数据保存在 sequence 中，在至多 SIMULATION_DEPTH 步探索结束之后，通过 REPTree 得到每一步的 Q 值，并将每一步的数据保存在 data 数组中，用于之后的 m_dataset 的随机更新（m_dataset 中的数据即为每次训练 REPTree 所用的数据）。到此 learnPolicy()函数结束。

在学习结束后，通过 getActionNoExplore()得到最优 action 并且返回。完成一次决策。

1.2 策略模型的表示、优点及缺点

策略模型使用了 epsilon greedy 的 Q-learning 方法。

优点在于模型简单，易于分析和训练，使用了 experience replay 使得训练 REPTree 的数据需求量少，数据利用率高。

缺点在于ϵ没有随着训练轮数的增加减小，导致在 Q 表收敛后，也可能采取较差的行动；Q-learning 本身存在过高估计的问题，在每一步中，都是选取 max 导致当前行动的估值略高于实际值。

1.3 变量分析

SIMULATION_DEPTH 是模拟的最大深度限制，对当前状态最多向前看 SIMULATION_DEPTH 步，这个变量使得，学习过程中考虑到未来可以得到的奖励，但又不会由于状态空间过多，无限搜索下去。

m_gamma 即为 Q 值更新公式中的γ，是未来奖励的衰减因子，表示未来奖励的重要程度，γ越大，则对未来的的奖励越重视，γ越小则越重视当前的奖励。

m_maxPoolSize 控制训练 REPTree 所用数据的最大数量，即经验池的大小，每一次学习得到的数据保存在 m_dataset 中，用于经验回放，如果 m_dataset 的数量超过了 m_maxPoolSize，则随机删除其中的数据，直

到大小为 m_maxPoolSize。

1.4 函数分析

getAction()函数多了下面几行代码，作用是以 m_epsilon 的概率随机选择 action，以 1-m_epsilon 的概率执行当前最优 action。

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

所以 getAction()是采用 epsilon greedy 策略选择 action，而 getActionNoExplore()则是一定会选择 Q 值最优的行动。

getAction()用于行为策略（Behavior Policy），即在训练过程中做决策。在探索 simulate()中，每一步使用带 epsilon greedy 的 getAction()进行探索，避免陷入局部最优解，并且不断的更新目标策略。这里与一般的 Q-learning 不同，一般在模拟时会使用 greedy 策略，每次必定选取最大值，这里却在探索中使用 epsilon greedy 的方法，对探索和利用进行折中。

getActionNoExplore()用于目标策略（Target Policy），即训练完毕后拿去应用的策略。在学习完成后，直接选择 Q 值最优的解作为真实的行动，使得当前行动最优。

2 修改特征提取方式

2.1 对游戏中属性的说明

经过打印中间结果，得到游戏属性信息，因为需要在代码中使用，对重要的信息在下面进行说明：

getImmovablePositions()可以得到砖块  和障碍物  （这里不是吃到可以得分的那个绿块）的信息，并且砖块的 itype 为 0，障碍物的 itype 为 13。

getMovablePositions()可以得到列车的信息，红色和橙色的列车 itype 为 7,8,10,11。

getPortalsPositions()可以得到得分块的信息，最上方吃到可以得分的绿色块 itype 为 4。

2.2 对原来代码的简单修改

原本 Pacman 的地图较大，为 28*31，但 freeway 中地图较小，但在移植过程中没有进行修改，在代码中 map 大小修改为 28*15。getAvatarSpeed()在全程其实没有发生变化，直接删除，getNPCPositions 的属性也全程为 null，直接删除相应代码。

2.3 添加代码

2.3.1 对位置的分析

添加了自身的 x 和 y 坐标，正上方三格以内是否有障碍物是否有障碍物（isTopBlock），在上方的一个小范围内是否有车（isDangerous），如果有，则在向上移动的过程中很可能被装到。自身与得分块的曼哈顿距离（目标块的位置通过 getPortalsPositions()得到），相关代码如下：

2.3.1.1 初始化

首先是对相关变量的初始化

```
boolean isTopBlock = false;
boolean isDangerous = false;
double Manhattan = 0;
```

2.3.1.2 代码实现

下面的代码首先通过 getAvatarPosition()得到自身的位置坐标，并进行离散化处理，然后遍历之前得到的所有物体，（对 itype 表示的含义在前面已经说明）第一个 if 判断正上方 3 格以内是否有障碍物，第二个判断是否在 x 坐标距离为 4，y 坐标上 2 格以内有车，最后一个 if 计算和得分块的曼哈顿距离。

```
Vector2d pos = obs.getAvatarPosition();
pos.x /= 28;
pos.y /= 28;

for(Observation o : allobj){
    Vector2d p = o.position;
    int x = (int)(p.x/28);
    int y= (int)(p.y/28);
    map[x][y] = o.itype;
    if(o.itype == 13 && x == pos.x && y-pos.y>=-3 && y-pos.y <=-1)
        isTopBlock = true;
    else if( (o.itype == 7 || o.itype == 8 || o.itype == 10 || o.itype == 11) &&
            Math.abs(x - pos.x) < 4 && y-pos.y>=-2 && y-pos.y <=0 )
        isDangerous = true;
    else if(o.itype == 4)
        Manhattan = Math.abs(x-pos.x) + Math.abs(y-pos.y);
}
```

2.3.1.3 其他相关代码

将得到的属性添加在 feature 数组中，在 datasetHeader 中添加相应的变量头

```
feature[420] = pos.x;
feature[421] = pos.y;
feature[422] = isTopBlock ? 1 : 0;
feature[423] = isDangerous ? 1 : 0;
feature[424] = Manhattan;
```

```
Attribute att = new Attribute( attributeName: "Pos_x" ); attInfo.addElement(att);
att = new Attribute( attributeName: "Pos_y" ); attInfo.addElement(att);
att = new Attribute( attributeName: "isTopBlock" ); attInfo.addElement(att);
att = new Attribute( attributeName: "isDangerous" ); attInfo.addElement(att);
att = new Attribute( attributeName: "Manhattan" ); attInfo.addElement(att);
```

2.3.2 调用自带函数的分析

利用 obs 的方法，对游戏的分数，时间，agent 的生命值，类型进行记录，经过测试，这些属性在游戏中都正常变化。

```
feature[425] = obs.getGameScore();
feature[426] = obs.getGameTick();
feature[427] = obs.getAvatarHealthPoints();
feature[428] = obs.getAvatarType();
```

```
att = new Attribute( attributeName: "GameScore" ); attInfo.addElement(att);
att = new Attribute( attributeName: "GameTick" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarHealthPoints" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarType" ); attInfo.addElement(att);
```

2.3.3 Feature 大小的调整

最终 map 数组大小为 28*15，并且添加了自身的 xy 坐标，isTopBlock，isDangerous，曼哈顿距离，四个调用自带函数的值，action 和 reward，最终将 feature 数组的大小调整为 432。

```
double[] feature = new double[432];
```

2.4 结果

在修改完特征提取方式后，观察到 agent 能够不断向上移动，比改进之前只是一味向右下角移动有所改进，但其他方面并没有观察到明显的改进，当前效果还是很差。希望在修改完强化学习的参数后，可以观察到明显的改进。

3 修改强化学习参数

3.1 epsilon

首先对 epsilon 进行修改，调整到 0.1 并且在 epsilon 大于 0.01 时，随着训练轮数的增多，getAction()函数调用时不断减小。

```
protected double m_epsilon=0.1;
protected double final_epsilon = 0.01;
public int getAction(double[] feature) throws Exception{
    if(m_epsilon > final_epsilon)
        m_epsilon -= (m_epsilon-final_epsilon)/100000;
```

3.2 SIMULATION_DEPTH和m_gamma

这两个参数是对未来情况的预测有关的参数

SIMULATION_DEPTH 越大，对未来预测的越远，m_gamma 越大，对未来的奖励越重视。

```
private static int SIMULATION_DEPTH = 1000;
```

```
protected double m_gamma = 0.9;
```

3.3 m_maxPoolSize

增大 m_maxPoolSize 的值，使得每次生成的 REPTree 有更多的训练集。但这个值过大时，游戏运行过慢，所以只是设置成了 2000。

```
protected int m_maxPoolSize = 2000;
```

3.4 其他尝试

多次测试后发现，小球总是向右移动，很多时候都是卡在右下角上下震荡，但没有找到问题所在。猜想是 REPTree 可能分类能力弱，不能很好地处理得到的 feature，尝试将其修改为 RandomForest，但会报错 weka.classifiers.trees.RandomTree: Cannot handle numeric class!，并没有成功使用。

修改为其他分类器，如 DecisionStump 也没有得到明显的改进。

3.5 实验结果

最终模型有了略微的改进，在少数情况下，可以移动到屏幕的中部，并且对列车有一定的躲避，但距离吃到目标块的差距还是很大，且大多数情况下，效果却很差，并且不断向右移动，卡在右边的角里。

我认为训练结果较差，一方面是因为训练的轮数极少，另一方面由于图像的信息比较复杂，只是存储通过手动提取是很难取得好的效果，图像存储在 map 中的信息可能并没有被 REPTree 充分的利用，如果将 REPTree 修改为 CNN，采用 DQN 的算法，经过多轮的训练，模型可能会得到一定的提升。