AlgorithmDog
freedom

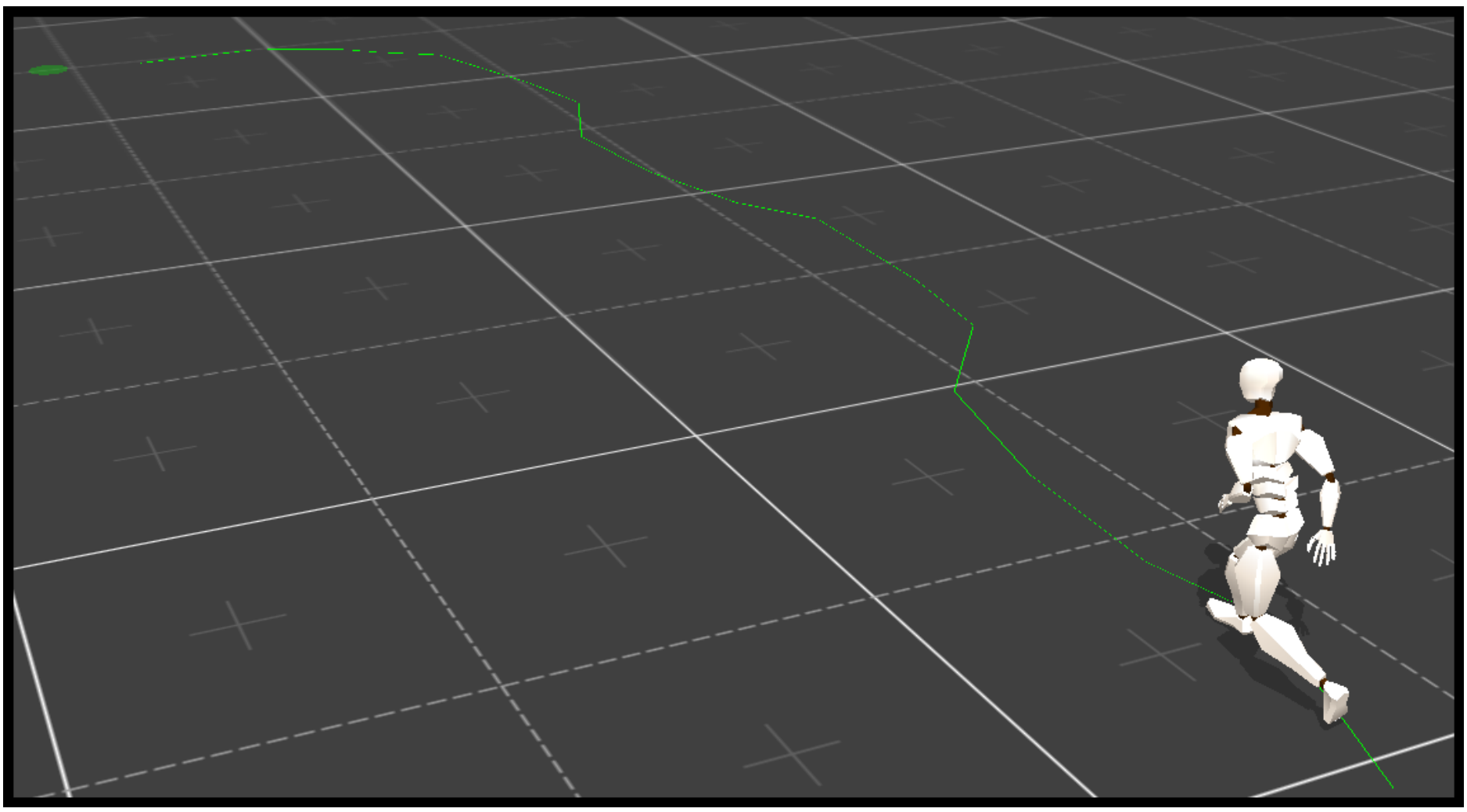
强化学习系列之五:价值函数近似

发表于2016年5月2日由lili

文章目录 [隐藏]

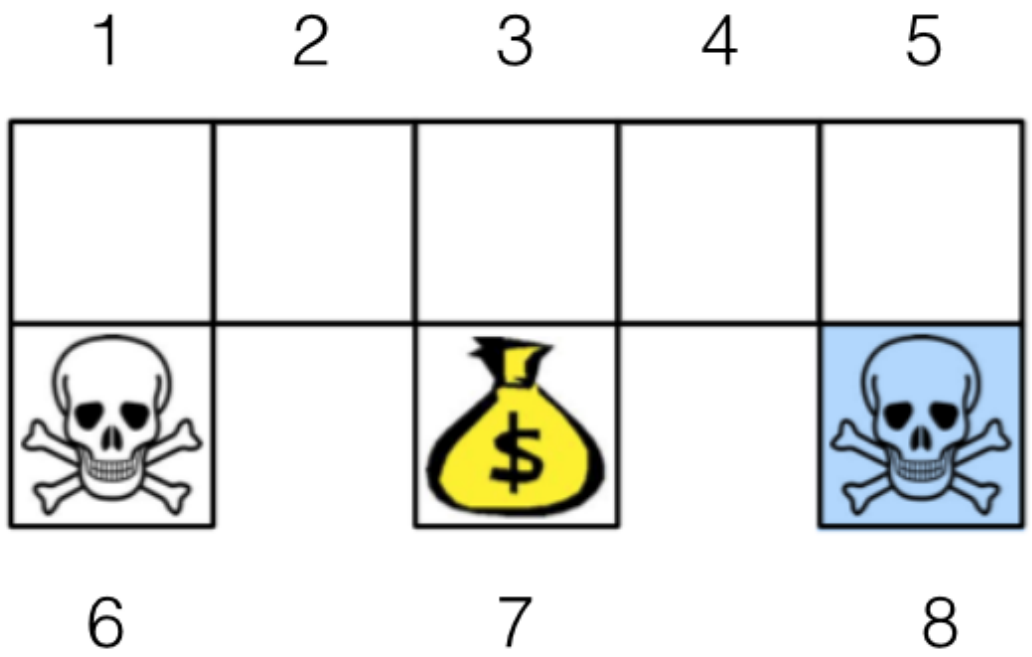
目前，我们已经介绍了一些强化学习的算法，但是我们无法在实际问题中运用这些算法。

为什么呢？因为算法估算价值函数 $v(s)$ 或者 $q(s,a)$，保存这些价值函数意味着保存所有状态。而实际问题中，状态的数目非常巨大，遍历一遍的事情就别想了。比如，围棋的状态总数是3^{19}，听说比宇宙的总原子数还多，23333。解决这个问题的方法是抽特征。对于一个状态 s, 我们抽取一些特征 $\hat{\boldsymbol{s}}$，将这些特征代替状态作为价值函数的输入，即 $v(\hat{\boldsymbol{s}})$ 或者 $q(\hat{\boldsymbol{s}},a)$。这种方法我们称之为价值函数近似。价值函数近似解决了海量状态之后，我们才能实用强化学习算法。



1. 价值函数参数化

我们又要以机器人找金币为场景介绍价值函数近似。机器人从任意一个状态出发寻找金币，找到金币则获得奖励 1，碰到海盗则损失 1。找到金币或者碰到海盗，机器人都停止。衰减因子 γ 设为 0.8。



机器人找金币只有 9 个状态，但为了介绍价值函数近似，我们就假装状态非常多。我们以四个方向是否有墙作为状态特征，比如状态 1 的特征为 $\hat{s} = [1, 0, 0, 1]$, 分别表示北 (东、南、西) 方向有 (没有、没有、有) 墙。状态太多的情况下，模型无关的强化学习算法比较有用。模型无关的强化学习算法的工作对象是 $q(s, a)$ (有状态特征之后为 $q(\hat{s}, a)$), 因此只有状态的特征是不够的。为此我们设定 $\boldsymbol{f(\hat{s}, a)}$ 特征向量一共分为 |A| 部分，分别对应不同的动作。在 $\boldsymbol{f(\hat{s}, a)}$ 特征向量, a 动作部分放上 $\hat{s}$ 特征，其他动作部分全部置为 0。比如机器人找金币场景，状态 1 采取向北动作的特征向量 $\boldsymbol{f(1,'n')}$ 如下。

$$\boldsymbol{f(1,'n')} = [\underbrace{1,0,0,1}_{a='n'},\underbrace{0,0,0,0}_{a='e'},\underbrace{0,0,0,0}_{a='s'},\underbrace{0,0,0,0}_{a='w'}]^T$$

搞出特征来了，接下来就用参数计算价值了。我们设定参数向量$\boldsymbol{w}$，然后用特征向量和权重向量的内积估计状态-动作价值。

$$q(\hat{s}, a) = \boldsymbol{f(\hat{s}, a)}^T \boldsymbol{w}$$

这时强化学习其实就是学习参数 $\boldsymbol{w}$ 的值，使得参数化的 q 值 $q(\hat{s}, a)$ 尽量接近最优策略的 q 值 $q^*(s, a)$，优化目标如下所示。

$$J(\boldsymbol{w}) = min \sum_{s \in S, a \in A} \{q(\hat{s}, a) - q^*(s, a)\}^2$$

我们用梯度下降法求解这个优化目标。梯度下降法首先要计算梯度 $\frac{\partial J}{\partial \boldsymbol{w}}$。直接求导可得梯度。

$$\frac{\partial J}{\partial \boldsymbol{w}} = \sum_{s \in S, a \in A} \{q(\hat{s}, a) - q^*(s, a)\} \boldsymbol{f(\hat{s}, a)}$$

但是状态很多，我们不可能真的按照上面的公式计算梯度 (上面的公式得遍历所有的状态)。实际的方法是让系统探索环境，遇到状态特征 $\hat{s}$ 和采取动作 a, 计算梯度然后更新参数。这个类似随机梯度下降。

$$\frac{\partial J}{\partial \boldsymbol{w}}_{\hat{s}, a} = \{q(\hat{s}, a) - q^*(s, a)\} \boldsymbol{f(\hat{s}, a)}$$

$$\Delta \boldsymbol{w} = -\alpha \frac{\partial J}{\partial \boldsymbol{w}}_{\hat{s}, a}$$

参数更新的代码如下所示。

```
#qfunc 是最优策略的 q 值
#alpha 是学习率
def update(policy, f, a, qfunc, alpha):
    pvalue        = policy.qfunc(f, a);
    error         = pvalue - tvalue;
    fea           = policy.get_fea_vec(f, a);
    policy.theta -= alpha * error * fea;
```

2. 强化学习算法

看了上面，你可以会问。计算 $\frac{\partial J}{\partial \boldsymbol{w}}_{\hat{s},a}$ 需要最优策略的 q 值，那我们上哪里去找这个值呢? 这是就是该强化学习算法上场了。我们回想一下三个模型无关的强化学习算法，都是让系统探索环境，探索时更新状态-动作价值。在更新时，MC Control 认为该样本的预期收益 g_t 为最优策略的 q 值，让状态-动作价值 q(s,a) 尽量接近。SARSA 认为 $r+\gamma q(s,a)$ 为最优策略的 q 值，Q Learning 认为 $r+argmax_{a'}\{\gamma q(s',a')\}$。

$$
\begin{aligned}
qfunc &= g_t & MCControl \\
qfunc &= r+\gamma q(\hat{s},a) & SARSA \\
qfunc &= r+max_{a'}\{\gamma q(\hat{s}',a')\} & QLearning
\end{aligned}
$$

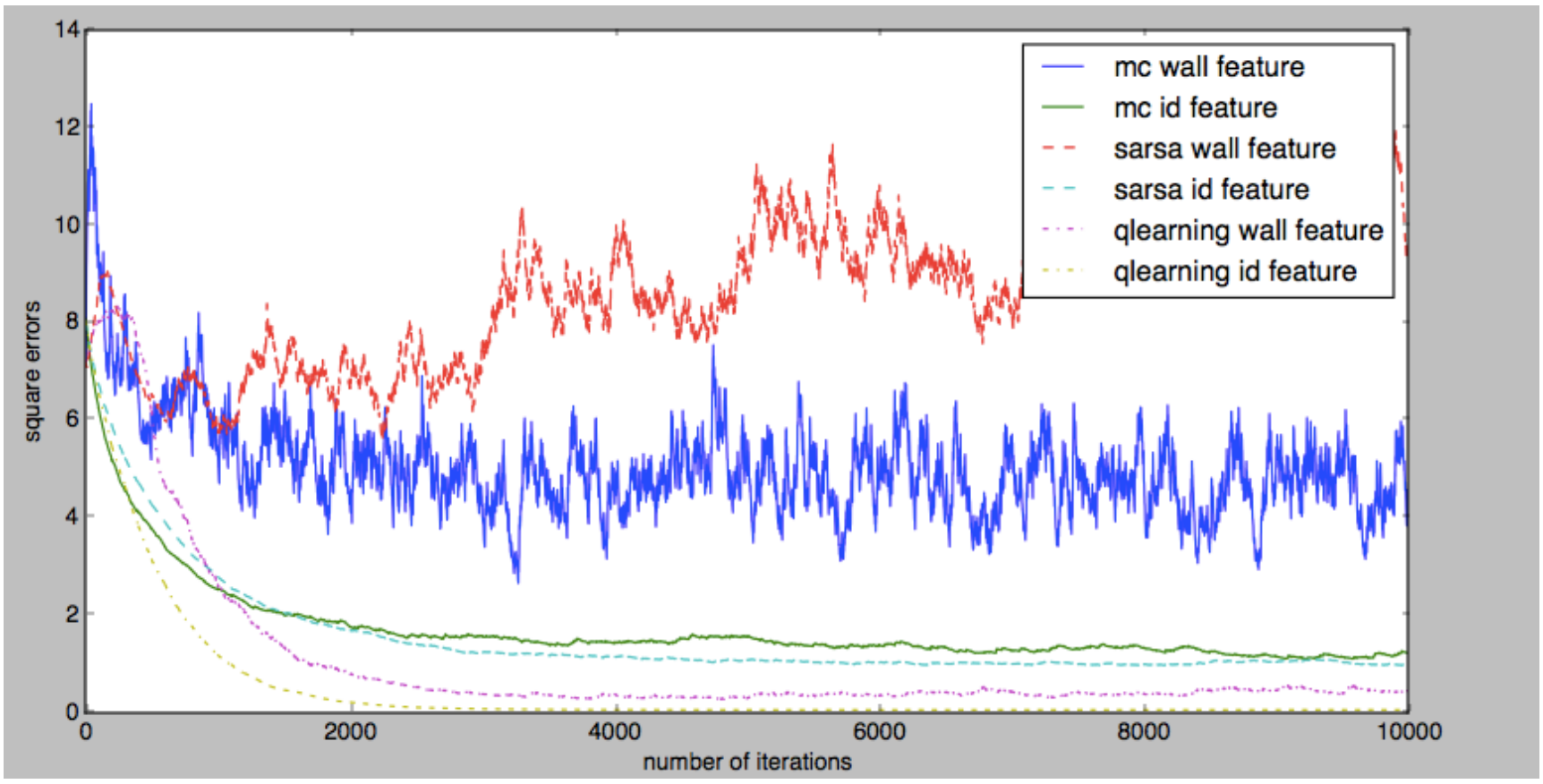
有了这些想法，我们只需要简单地改变下强化学习算法的更新部分，就可以引入价值函数近似了。新的更新规则是将算法认为的最优策略的 q 值输入参数更新模块。觉个例子，价值函数近似之后的 Q Learning 算法代码如下所示。

```
def qlearning(grid, policy, num_iter1, alpha):
    actions = grid.actions;
    gamma   = grid.gamma;
    for i in xrange(len(policy.theta)):
        policy.theta[i] = 0.1

    for iter1 in xrange(num_iter1):
        f = grid.start();
        #从一个随机非终止状态开始， f 是该状态的特征
        a = actions[int(random.random() * len(actions))]
        t = False
        count = 0

        while False == t and count < 100:
            t,f1,r       = grid.receive(a)
            #t  表示是否进入终止状态
            #f1 是环境接受到动作 a 之后转移到的状态的特征。
            #r  表示奖励

            qmax = -1.0
            for a1 in actions:
                pvalue = policy.qfunc(f1, a1);
                if qmax < pvalue:  qmax = pvalue;
            update(policy, f, a, r + gamma * qmax, alpha);

            f            = f1
            a            = policy.epsilon_greedy(f)
            count       += 1
    return policy;
```

3. 做个实验

我们用机器人找金币做个实验吧。实验中，我们用了两种特征。一种特征是强特征，也就是上述四个方向是否有墙特征。另一种特征是 id 特征，特征向量长度为状态个数，第 i 个状态的特征向量的第 i 位为 1，其他位置为 0。实验对比了三种算法: MC Control, SARSA 和 Q Learning。$\epsilon-$贪婪策略的 ϵ 设为 0.2， 学习率α 设为 0.001。

和上文一样，算法计算得到的状态(特征)-动作价值和最优策略的状态-动作价值之间的平方差，当做评价指标。实验的结果如下图所示。



这个实验结果告诉我们的第一件事就是选好特征。墙特征比 id 特征差。状态2 和 4 都是南北方向有墙，墙特征是一样的，会造成混淆。id 特征就没有这个问题。表现在实验结果上，MC Control 和 SARSA 在墙特征上都不停震荡，同时三种算法在墙特征的表现都不如其在 id 特征上的表现。

Q Learning 一如既往地表现出优越的效果。在墙特征上， Q Learning 不仅没有像 MC Control 和 SARSA 一样震荡，而且效果远远好于它们两者。在 id 特征上， Q Learning 完美拟合了最优策略的状态-动作价值。

4. 总结

实际问题中，状态的数目非常多，因此基于状态-动作价值的强化学习算法不适用。为了解决这个问题，人们提出了价值函数近似的方法。价值函数近似用特征表示状态或者状态-动作，用参数向量计算价值。价值近似之后，我们才算能把强化学习算法应用在实际问题上。本文代码可以在 Github 上找到，欢迎有兴趣的同学帮我挑挑毛病。强化学习系列的下一篇文章将介绍基于梯度的强化学习。

文章结尾欢迎关注我的公众号 AlgorithmDog，每周日的更新就会有提醒哦~

欢迎关注

公众号讲述**机器学习**和**系统研发**的轶事，
希望讲得有趣，**每周日更新**~

扫描二维码即可关注。您，不关注下么?

强化学习系列系列文章

- 强化学习系列之一:马尔科夫决策过程
- 强化学习系列之二:模型相关的强化学习
- 强化学习系列之三:模型无关的策略评价
- 强化学习系列之四:模型无关的策略学习
- 强化学习系列之五:价值函数近似

- 强化学习系列之六:策略梯度
- 强化学习系列之九:Deep Q Network (DQN)

此条目发表在强化学习, 算法荟萃分类目录，贴了强化学习标签。将固定链接加入收藏夹。

《强化学习系列之五:价值函数近似》有 11 条评论

圈儿 说:
2016年5月16日下午6:18

楼主，在五的GitHub中，你的“experiment.py” 没有上传。能上传一下吗?

回复

上微博的猫V说:
2016年5月16日下午10:26

sorry，我代码在另一台电脑上，明天传。谢谢您指出。

回复

上微博的猫V说:
2016年5月18日下午4:04

已经上传，再次感谢～。

回复

Pingback引用通告： 强化学习系列之七:在 OpenAI Gym 上实现 QLearning | AlgorithmDog

小小蚂蚁 说:
2016年6月29日下午8:51

我想问问大神您实验的开发环境可以是windows吗?

回复

上微博的猫V说:
2016年7月1日下午7:17

是mac哦。

回复

Lebo 说:
2016年7月5日下午5:12

lz有遇到Qlearning某种feature不收敛的情况吗？该如何处理?

回复

Lebo 说:
2016年7月5日下午9:09

应该是SARSA和Qlearning都不收敛，可能是验证了TD算法Linear Feature可能不收敛，但对于Table Lookup，所有方法都收敛

回复

上微博的猫V说:

2016年9月4日下午3:14

恩恩，Linear Feature 上 SARSA 和 Q Learning确实是不收敛的，我也在查资料看下有没有好的办法

回复

LemonMan*说:*

2016年9月3日下午7:09

你好，围棋的状态总数应该是3的361次方

回复

Elitack*说:*

2017年4月15日下午3:09

想说下代码部分update函数应该是这样子吧:

```
def update(policy, f, a, tvalue, alpha):
pvalue = policy.qfunc(f, a);
error = pvalue - tvalue;
fea = policy.get_fea_vec(f, a);
policy.theta -= alpha * error * fea;
```

其中tvalue 应该是最优策略

然后想说的是github中好像文件grid_map不见了，希望可以补充一下～

回复
