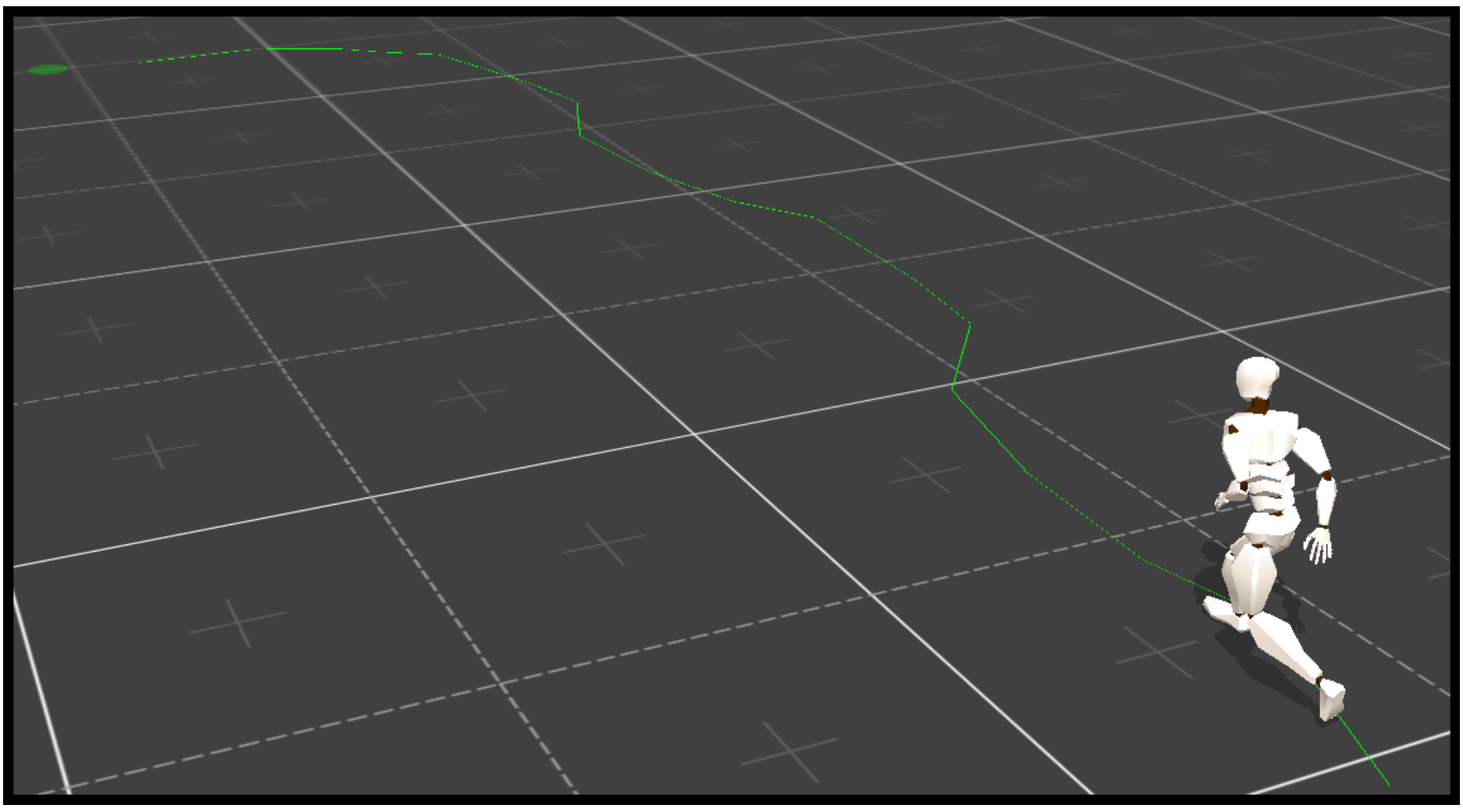
AlgorithmDog
freedom

强化学习系列之四:模型无关的策略学习

发表于2016年4月24日由lili

文章目录 [隐藏]

模型无关的策略学习，是在不知道马尔科夫决策过程的情况下学习到最优策略。模型无关的策略学习主要有三种算法: MC Control, SARSA 和 Q learning。



1. 一些前置话题

在模型相关强化学习中，我们的工作是找到最优策略的状态价值$\boldsymbol{v}$。但是在模型无关的环境下，这个做法却行不通。如果我们在模型无关环境下找最优策略的状态价值$\boldsymbol{v}$，在预测时，对状态 $\boldsymbol{s}$ 最优策略如下所示。

$$\pi(s,a)=\begin{cases}1 & a=argmax_a R_{s,a}+\gamma\sum_{s'\in S}T_{s,a}^{s'}v(s') \\ 0 & a\neq argmax_a R_{s,a}+\gamma\sum_{s'\in S}T_{s,a}^{s'}v(s')\end{cases}$$

同学们看到 $R_{s,a}$ 和 $T_{s,a}$ 了没？在模型无关的设定下，我们不知道这两个值。也许有同学说可以在预测时探索环境得到 $R_{s,a}$ 和 $T_{s,a}$。但是实际问题中，探索会破坏当前状态。比如机器人行走任务中，为了探索机器人需要做出一个动作，这个动作使得机器人状态发生变化。这时候为原来状态选择最优动作已经没有意义了。解决办法是把工作对象换成状态-动作价值 $\boldsymbol{q}$。获得最优策略的状态-动作价值 $\boldsymbol{q}$ 之后，对于状态 $\boldsymbol{s}$ 最优策略如下所示。

$$\pi(s,a) = \begin{cases} 1 & a = argmax_a q(s,a) \\ 0 & a \neq argmax_a q(s,a) \end{cases}$$

另一个话题关于贪婪策略。最优策略是贪婪策略，这个不用怀疑。但对于学习来说，贪婪策略不一定是最好的。举个栗子，状态集合为 {A, B, C, D}， 其中 A 是起始状态而 D 是终止状态。A->B 奖励为1 而 C->D 奖励为100，其他奖励为0。从起始状态 A, 贪婪策略会一直选择进入 B，而不探索 C 从而获得更高奖励。为了鼓励探索，我们用了另一种$\epsilon-$贪婪策略，其公式如下。

$$\pi_{\epsilon-greedy}(s,a) = \begin{cases} 1-\epsilon+\frac{\epsilon}{|A|} & a = argmax_a q(s,a) \\ \frac{\epsilon}{|A|} & a \neq argmax_a q(s,a) \end{cases}$$

2. MC Control

一听到 Monte Carlo Control (MC Control) 这个名字，我们就知道，这个算法生成样本然后根据样本计算状态-动作价值。对于每一个状态-动作对，我们都要维持他们的价值 $q(s,a)$ 和被访问次数 $n(s,a)$。让系统采样一个状态-动作-奖励的系列，然后对于每个状态-动作更新价值和次数。

$$q(s,a) = \frac{q(s,a) * n(s,a) + g}{n(s,a) + 1}$$
$$n(s,a) = n(s,a) + 1$$

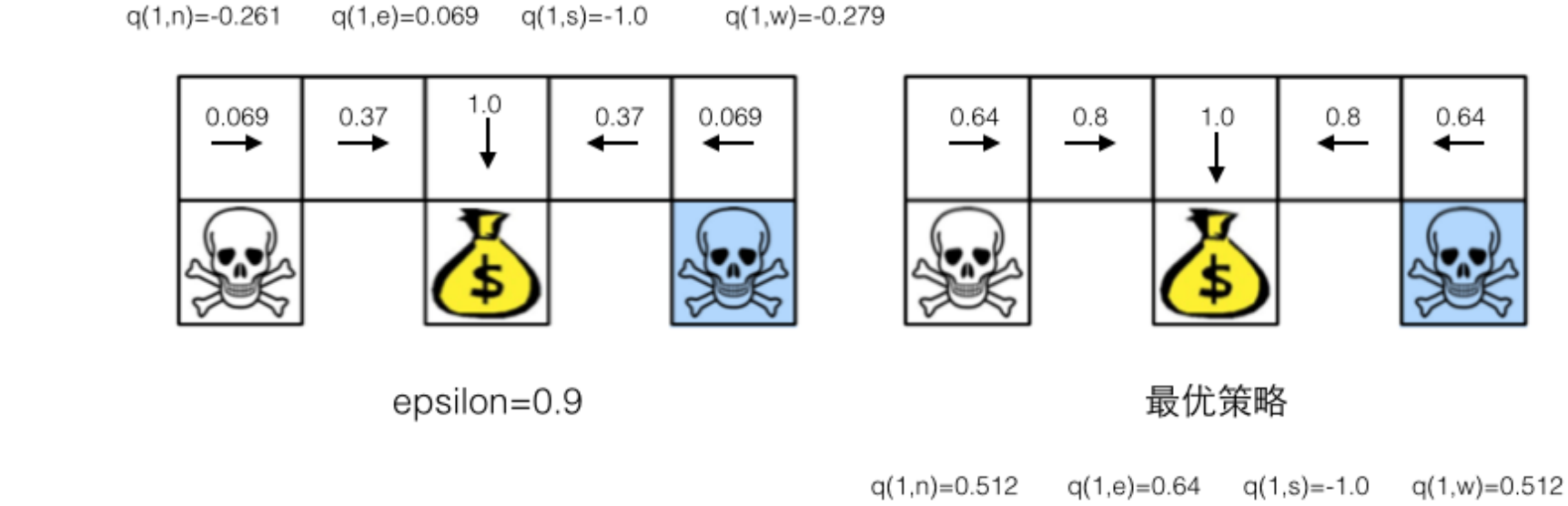
其中$g = r_t + \gamma r_{t+1} + ...$是预期衰减奖励之和。MC Control 算法的代码如下。

```
def mc(num_iter1, epsilon):
    n = dict();
    for s in states:
        for a in actions:
            qfunc["%d_%s"%(s,a)] = 0.0
            n["%d_%s"%(s,a)]     = 0.001 //平滑

    for iter1 in xrange(num_iter1):
        s_sample = []
        a_sample = []
        r_sample = []

        s = states[int(random.random() * len(states))]
        t = False
        while False == t:
            a = epsilon_greedy(s, epsilon)
            t, s1, r  = grid.transform(s,a)
            s_sample.append(s)
            r_sample.append(r)
            a_sample.append(a)
            s = s1

        g = 0.0
        for i in xrange(len(s_sample)-1, -1, -1):
            g *= gamma
            g += r_sample[i];

        for i in xrange(len(s_sample)):
            key = "%d_%s"%(s_sample[i], a_sample[i])
```

```
n[key]      += 1.0;
qfunc[key]   = (qfunc[key] * (n[key]-1) + g) / n[key]

g -= r_sample[i];
g /= gamma;
```

在 MC Control 算法中，状态-动作价值会收敛到$\epsilon-$贪婪策略的状态-动作价值，不会收敛到最优策略的状态-动作价值。不过，这里有一个很好玩的事情。MC Control 采用$\epsilon-$贪婪策略算出了状态-动作价值 $\boldsymbol{q}$ 。如果在预测时采用贪婪策略，系统很有可能是遵循最优策略。

还是拿机器人找金币当例子。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。左边是 MC Control 采用 $\epsilon = 0.9$ 贪婪策略的 $\boldsymbol{q}$ (1 状态标示了所有的 q 值，其他状态只标示了最大的 q 值，其中)，0.9 是很大的 ϵ 表示策略有很大的随机性。在这个状态-动作价值采用贪婪策略，策略动作和右边最优策略采取的动作完全一致。



不知道这个论断理论上是不是成立的？如果有哪位大牛了解，期待您的指导。

3. SARSA

State Action Reward State Action (SARSA) 算法其实是状态-动作价值版本的时差学习 (Temporal Difference, TD) 算法。SARSA 利用马尔科夫性质，只利用了下一步信息。SARSA 让系统按照策略指引进行探索，在探索每一步都进行状态价值的更新，更新公式如下所示。

$$q(s,a) = q(s,a) + \alpha(r + \gamma q(s',a') - q(s,a)) \tag{1}$$

s 为当前状态，a 是当前采取的动作，s' 为下一步状态，a' 是下一个状态采取的动作，r 是系统获得的奖励，α 是学习率，γ 是衰减因子。SARSA 的代码如下。

```
def sarsa(num_iter1, alpha, epsilon):
    for s in states:
        for a in actions:
            key = "%d_%s"%(s,a)
            qfunc[key] = 0.0

    for iter1 in xrange(num_iter1):
        s = states[int(random.random() * len(states))]
        a = actions[int(random.random() * len(actions))]
        t = False
        while False == t:
            key         = "%d_%s"%(s,a)
            t,s1,r      = grid.transform(s,a)
            a1          = epsilon_greedy(s1, epsilon)
            key1        = "%d_%s"%(s1,a1)
```

```
qfunc[key]  = qfunc[key] + alpha * ( \
              r + gamma * qfunc[key1] - qfunc[key])
s           = s1
a           = a1
```

SARSA 收敛到哪里呢？和 MC Control 算法一样，SARSA 的状态-动作价值也收敛到 $\epsilon-$ 贪婪策略的状态-动作价值上。

4. Q Learning

Q Learning 的算法框架和 SARSA 类似。Q Learning 也是让系统按照策略指引进行探索，在探索每一步都进行状态价值的更新。关键在于 Q Learning 和 SARSA 的更新公式不一样，Q Learning 的更新公式如下。

$$q(s,a) = q(s,a) + \alpha\{r + max_{a'}\{\gamma q(s',a')\} - q(s,a)\} \tag{2}$$

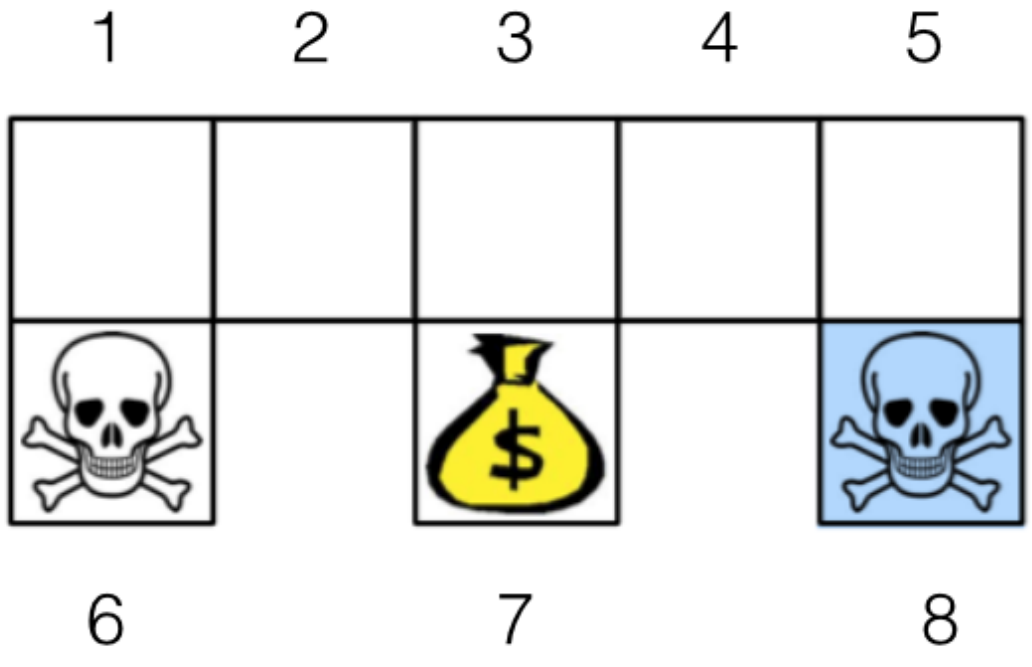
Q Learning 的代码如下。

```
def qlearning(num_iter1, alpha, epsilon):

    for s in states:
        for a in actions:
            key = "%d_%s"%(s,a)
            qfunc[key] = 0.0

    for iter1 in xrange(num_iter1):

        s = states[int(random.random() * len(states))]
        a = actions[int(random.random() * len(actions))]
        t = False
        while False == t:
            key         = "%d_%s"%(s,a)
            t,s1,r      = grid.transform(s,a)

            key1 = ""
            qmax = -1.0
            for a1 in actions:
                if qmax < qfunc["%d_%s"%(s1,a1)]:
                    qmax = qfunc["%d_%s"%(s1,a1)]
                    key1 = "%d_%s"%(s1,a1)
            qfunc[key]  = qfunc[key] + alpha * ( \
                          r + gamma * qfunc[key1] - qfunc[key])
            s           = s1
            a           = epsilon_greedy(s1, epsilon)
```

Q Learning 的收敛性是很好玩的。Q Learning 与 MC Control 和 SARSA 一样采用了 ϵ-贪婪策略，但 Q Learning 的状态-动作价值却能收敛到最优策略的状态-动作价值。

5. 做点实验

实验还是以机器人找金币为场景。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。



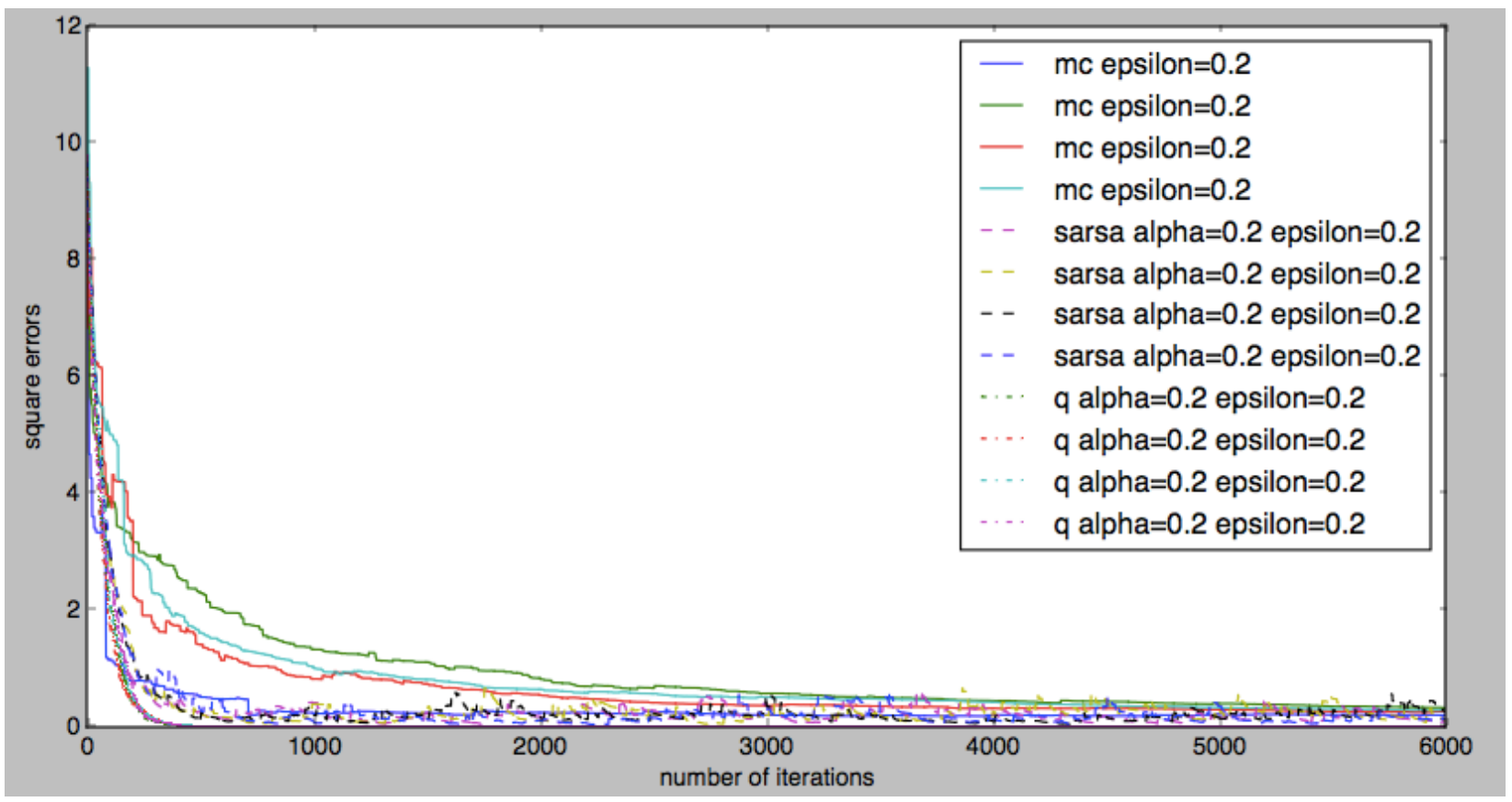
我们将算法计算得到的状态-动作价值和最优策略的状态-动作价值之间的平方差，作为评价指标，其计算公式如下。

$$square-error=\sum_{s\in S,a\in A}\{q(s,a)-q^*(s,a)\}^2 \tag{3}$$

其中$q^*(s,a)$是最优策略的状态-动作价值。

5.1. 算法稳定性

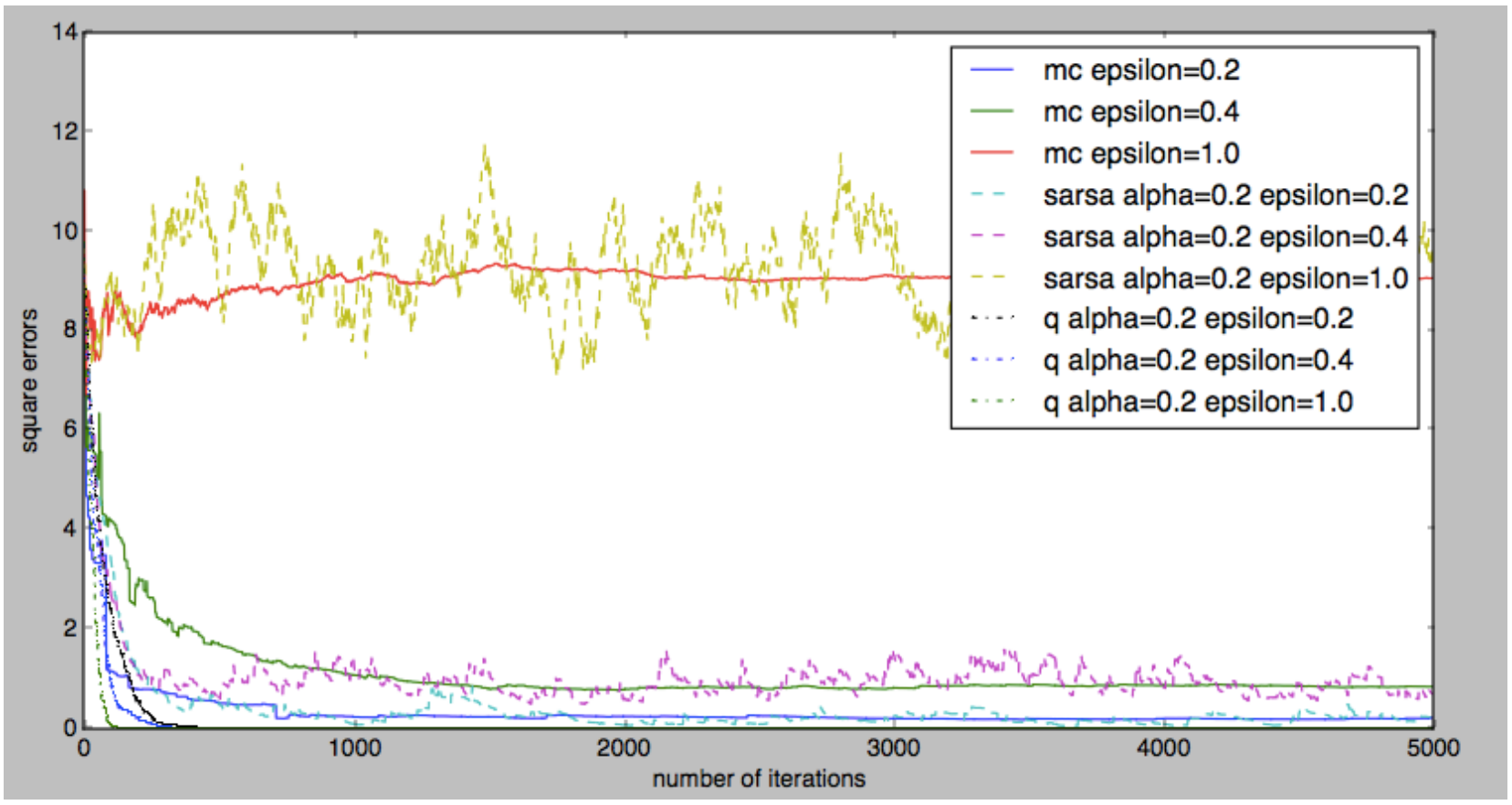
MC Control、 SARSA 和 Q Learning 在算法运行过程中，都有随机因素。我们会关心每次运行的效果是类似的还是差别很大，也就是算法的稳定性。从下图我们可以看到，MC Control 是最不稳定的算法。平方误差下降阶段，SARSA 的稳定性很好，但收敛之后 SARSA 会上下抖动。 Q Learning 拥有良好的稳定性。



5.2. $\epsilon-$贪婪策略的影响

ϵ 对算法有影响。ϵ 最大为 1 的时候，MC Control 和 SARSA 的平方误差很大，Q Learning 能够让平方误差降到 0。其实这个时候$\epsilon-$ 贪婪策略相当于随机策略。MC Control 和 SARSA 的状态-动作价值收敛到了随机策略的状态-动作价值，因此保持一个比较大的值。Q Learning 依然能够收敛到最优策略的状态-动作价值，因此能降到 0。随着 ϵ 的下降，MC Control 和 SARSA 收敛之后的平方误差会降低，Q Learning 则一如既往地降到 0。

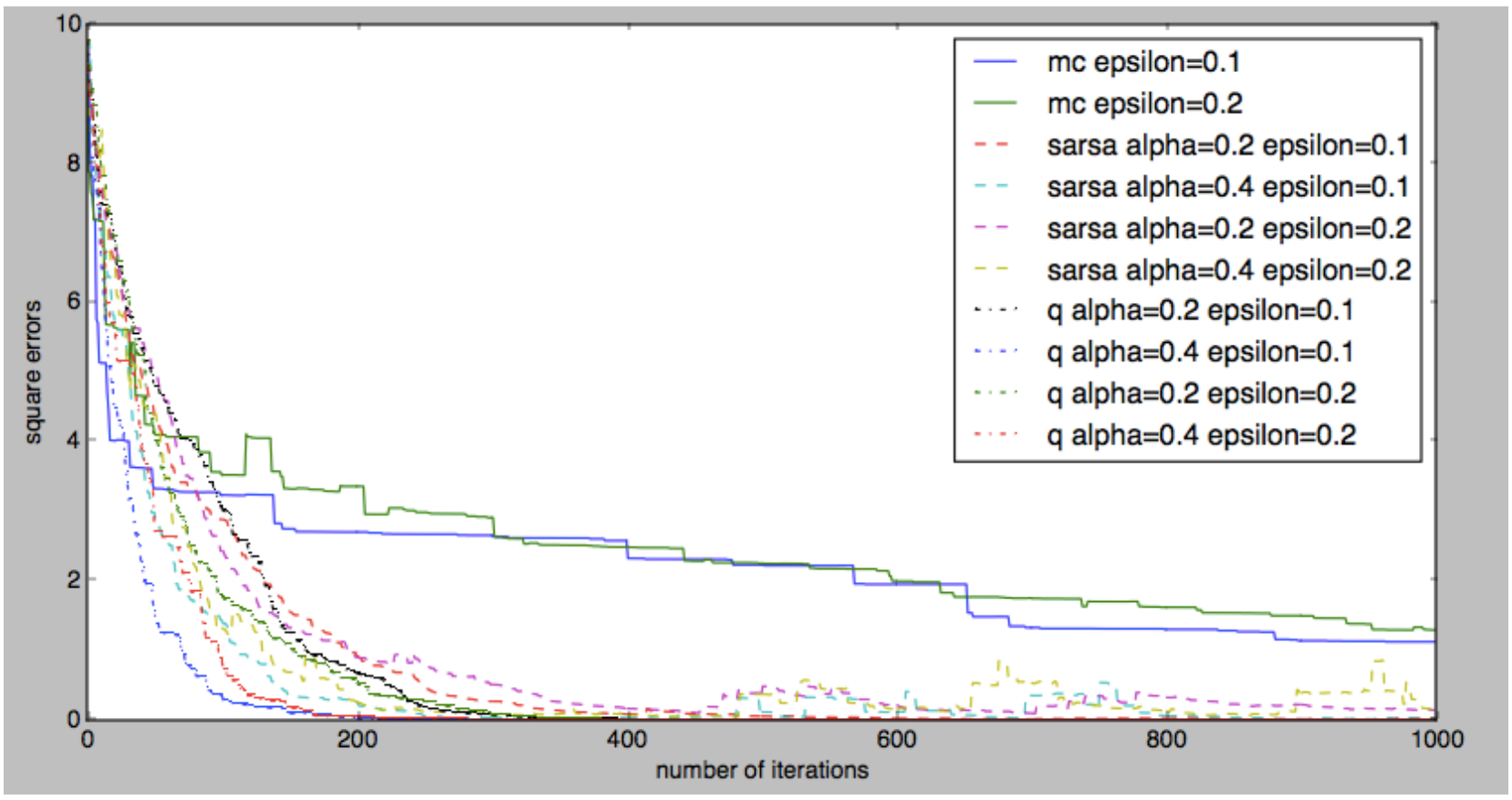
这个实验表明 Q learning 能从其他策略探索经验中学习。我们称能够从其他策略学习到最优策略的算法为离策略 (off-policy) 算法，反之为在策略 (on-policy) 算法。MC Control 和 SARSA 是在策略的，Q Learning 是离策略的。



还有一点就是，ϵ 越大, SARSA 收敛之后抖动就越厉害。

5.3. 不同算法的效果对比

全面考察这三种算法，在机器人找金币这个场景上，Q Learning 要好于 SARSA，SARSA 要好于 MC Control。



6. 总结

本文介绍了模型无关的策略学习。模型无关的策略学习主要有三种算法: Monte Carlo Control, Sarsa 和 Q learning。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~

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。

《强化学习系列之四:模型无关的策略学习》有 9 条评论

Not_GOD 说:
2016年5月5日下午3:11

$pi_epsilon$−greedy(s,a) 定义是不是 缺了 $1−epsilon+frac{epsilon}{|A|}$

回复

上微博的猫V 说:
2016年5月5日下午3:34

确实。已经改正，谢谢～

回复

Not_GOD 说:
2016年5月5日下午5:07

公式(2) 不应该使用 argmax 而应该是 max 哈，check一下

回复

上微博的猫V 说:
2016年5月5日下午5:18

公式（2）是指第二个公式嘛？如果是，应该使用 (argmax) 吧。(pi(s,a)) 当 a 是价值最大的动作时，等于1；不然等于0。

回复

学名马铃薯 说:

2016年5月6日上午11:14

感觉Q learning的公式有点问题，在我理解中argmax_{a} f(a), 是取f(a)最大值中a的值，而程序大概意思是gamma*f(argmax_{a} f(a))，感觉可以直接用qmax

回复

上微博的猫V说:

2016年5月6日上午11:23

公式已经改成max啦。程序用确实qmax比较简便，我的程序比较烦。

回复

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

Steve说:

2016年11月10日下午2:11

您好，其中與模型相關的Q-Learning等剛好都是我在搜尋的資料。很謝謝您的分享讓我有初步的認識。但其中仍很多相關資料看不是很懂，請問對於初學者您友推薦什麼書或資源等方法供我比較好入門呢?謝謝您

回复

lili说:

2016年11月19日下午9:05

David Silver 的课可以看下。

回复

AlgorithmDog

自豪地采用WordPress。