Policy invariance under reward transformations: Theory and application to reward shaping

Andrew Y. Ng, Daishi Harada, Stuart Russell
Computer Science Division
University of California, Berkeley
Berkeley CA 94720
{ang,daishi,russell}@cs.berkeley.edu

【强化学习思想 22】Reward Shaping Invariance

张楚珩
清华大学 交叉信息院博士在读

原文传送门

Ng, Andrew Y., Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping." ICML. Vol. 99. 1999.

特色

个人感觉遇到实际的强化学习问题的时候，如果想解决地更好，与其上一大堆高端的方法，不如把reward engineering做好。奖励就好比强化学习算法的指路标，如果奖励设计得好，就很容易一步步地引导算法收敛到正确的位置。那么什么是好的奖励，如何把一个“不好”的奖励通过一些变形变成“好”的奖励呢？这篇90后的老文章给出了简洁而深刻的解答。

过程

对于一个MDP $M=(S,A,T,\gamma,R)$ ，考虑对这个MDP的奖励函数进行变形 $R'=R+F$ ，得到新的一个MDP $M'=(S,A,T,\gamma,R')$ 。通过解新的这个MDP来得到原来MDP的解。研究的问题是对于怎样的变形能保证 M' 中得到的最优策略仍然是 M 中的最优策略。

文章告诉我们，叠加的奖励满足 $F(s,a,s')=\gamma\Phi(s')-\Phi(s)$ 形式是最优策略不变的**充要条件**。具体表述如下（注意文章里面的 s_0 的absorbing state）

Theorem 1 *Let any S, A, γ, and any shaping reward function $F : S \times A \times S \mapsto \mathbb{R}$ be given. We say F is a* **potential-based** *shaping function if there exists a real-valued function $\Phi : S \mapsto \mathbb{R}$ such that for all $s \in S - \{s_0\}, a \in A, s' \in S$,*

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s), \tag{2}$$

(where $S - \{s_0\} = S$ if $\gamma < 1$). Then, that F is a potential-based shaping function is a necessary and sufficient condition for it to guarantee consistency with the optimal policy (when learning from $M' = (S, A, T, \gamma, R + F)$ rather than from $M = (S, A, T, \gamma, R)$), in the following sense:

- *(Sufficiency) If F is a potential-based shaping function, then every optimal policy in M' will also be an optimal policy in M (and vice versa).*
- *(Necessity) If F is not a potential-based shaping function (e.g. no such Φ exists satisfying Equation (2)), then there exist (proper) transition functions T and a reward function $R : S \times A \mapsto \mathbb{R}$, such that no optimal policy in M' is optimal in M.*

证明的过程十分简单，我们这里提几个重要的结论和应用：

1. 原MDP和新MDP价值函数的关系

$$
\begin{aligned}
Q^*_{M'}(s,a) &= Q^*_M(s,a) - \Phi(s), \\
V^*_{M'}(s) &= V^*_M(s) - \Phi(s).
\end{aligned}
$$

即新MDP中的（最优）价值函数用 $\Phi(s)$ 垫高一下就是原来MDP中的（最优）价值函数了；其实也不是必须是最优价值函数，对于任意策略的价值函数都有此性质。这意味着并不是只是收敛后的价值函数满足这样的关系，整个学习的过程中都是被这样的一个constant potential function垫高了，因此不会给学习到的过程带来不稳定。

2. 如果 $R(s,a,s') = \gamma\Phi(s') - \Phi(s)$ ，那么该MDP上任意策略都最优（最差）

这是说如果我们之间把这样基于势能的函数作为奖励函数的话，那么这个MDP变成了一个平的MDP，任意的策略都没有更好和更坏之分。这也再一次说明了为什么我们加这么一个奖励函数上去的时候，不会影响到最优策略。

3. Reward shaping如何加速算法收敛？

如果我们对于最优价值函数 $V^*_M(s)$ 有一个大致的认识，那么我们可以把势能设置为 $\Phi(s) = \hat{V}^*_M(s)$ ，并据此设计奖励函数 F 。注意到，如果估计的足够准，那么所需要学习的价值函数 $V^*_{M'}(s) \to 0$ 是一个很平坦的形态，很快就能够学习得到。

4. Reward shaping如何处理子目标（subgoal）？

本专栏讲了很多HRL的内容，子目标是HRL里面一个重要的概念，大致讲的是通过定义一些子目标来帮助agent找到通向最终目标的道路。可以定义这样一个奖励函数 F ，在agent**第一次**到达子目标的时候给予一定的奖励。可以想象这样的定义也是符合potential的定义的。这样的定义也能加速agent学习，不过显然它还不够好，比如它还不够dense，不过基于文章的理论，我们可以很容易对其改进。

充要性证明？

充分性，以下一个式子基本上就说明了

$$
Q^*_M(s,a) - \Phi(s) = \mathrm{E}_{s'}\Big[R(s,a,s') + \gamma\Phi(s') - \Phi(s) \\
+\gamma \max_{a' \in A}\left(Q^*_M(s',a') - \Phi(s')\right)\Big]
$$

必要性，大致上说的是如果 $F(s_1,s_2) \neq \gamma\Phi(s_2) - \Phi(s_1) = \gamma F(s_1,s_0) - F(s_2,s_0)$ ，那么就可以构造 T 和 R ，使得在 s_1 状态上时一个最优策略是选择 s_2 ，而另一个选择 s_2 。

一个更好懂的具体例子

文章里面的例子是10x10的格子看得我眼花，而且还是一个stochastic dynamics，这里举一个更简单的例子。一个10个格子的格子世界，开始在第1个格子，目标是第10个格子；两种行动，往左和往右，如果在边上就仍然在边上；到达第10个格子的时候收到1的奖励；infinite case with $\gamma = 0.9$ 。聪明的你应该看出来了，最优策略是直接往右走到第10个格子，然后每次再往右，这样还留在第10个格子，不停刷分。再聪明一点的话也看出来了，第10个格子上面的状态价值函数值应该为 $1/(1-\gamma) = 10$ 。

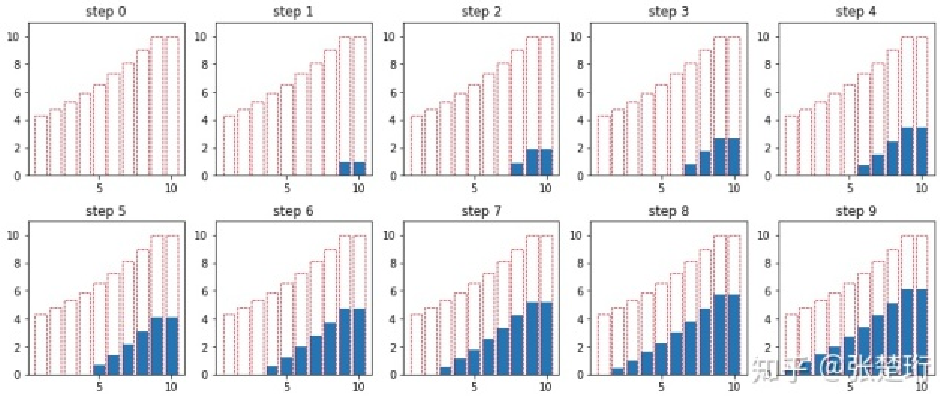
我们再简化一点，就只看状态价值函数V而不是Q，并使用Value Iteration来求解。

我们大致上猜得到价值函数应该从左往右越来越高，我们猜一个，比如

```
phi = np.array([[1], [1], [3], [3], [5], [5], [7], [7], [9], [9]])
```

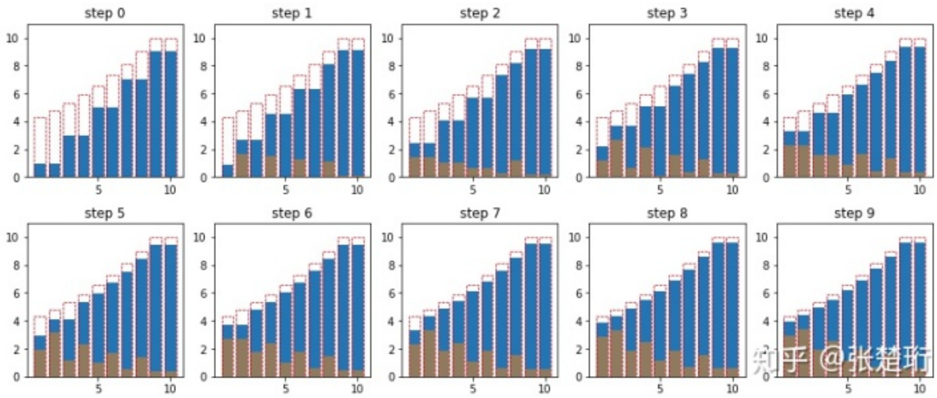
然后看看是不是收敛的更快。

没有使用reward shaping的情况



不使用reward shaping：红框代表最优价值函数，蓝条代表目前学习到的价值函数

再看看使用reward shaping的情况



使用reward shaping：红框代表最优价值函数，蓝条代表目前学习到的价值函数V_M' + Phi，其中橙条代表V_M'

可以看到，如果我们垫上去的势能如果足够准，那么需要学习的部分就没多少了，因此自然收敛更快。

想要代码么？

```python
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

# prefetched optimal state value function obtained from value iteration
v_opt = np.array([[ 4.3046721], [ 4.782969 ], [ 5.31441  ], [ 5.9049    ], [ 6.561
    [ 7.29      ], [ 8.1       ], [ 9.        ], [10.       ], [10.       ]])

# number of grids
n = 10

# transition probability of action moving left
# it is a deterministic dynamics
P1 = np.identity(n)[:-1]
P1 = np.concatenate([[P1[0]], P1])

# transition probability of action moving right
P2 = np.identity(n)[1:]
P2 = np.concatenate([P2, [P2[-1]]])

# initial state values
# you can try different initializations
# - it is proved that initialization is equavalent to potential reward shaping
v = np.zeros((n, 1))

# original reward function
r = np.zeros((n, 1))
r[-1] = 1

# discount rate
gamma = 0.9

# original Bellman operator
Bellman_op = lambda v: np.maximum(np.matmul(P1, r) + gamma * np.matmul(P1, v),
                                  np.matmul(P2, r) + gamma * np.matmul(P2, v))
# reward shaped Bellman operator
Bellman_op_rs = lambda v: np.maximum(np.matmul(P1, r) + gamma * np.matmul(P1, v),
                                  np.matmul(P2, r) + gamma * np.matmul(P2, v))

# value iteration and plot
plt.figure(figsize=(15, 6))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.bar(np.arange(n) + 1, (v + phi).flatten())
    plt.bar(np.arange(n) + 1, v.flatten(), alpha=0.5)
    plt.bar(np.arange(n) + 1, v_opt.flatten(), edgecolor='r',
            color='None', linewidth=0.75, linestyle='--')
    plt.ylim([0, 11])
    plt.title('step {}'.format(i))
    v = Bellman_op_rs(v)
plt.subplots_adjust(hspace = 0.3)
```

编辑于 2019-05-24

机器学习 强化学习 (Reinforcement Learning)

文章被以下专栏收录

强化学习前沿
读呀读paper