Grid World

刘经宇

2023 年 6 月 30 日

这篇报告是关于 Sutton 和 Barto 的著作 <Reinforcement Learning> 中例 3.5 和例 3.8 grid world 的.

1 问题介绍

我们考虑图 1 所示的 grid world. 每个格点是一个状态, 在每个点上, 有 4 个可选的动作: 东,

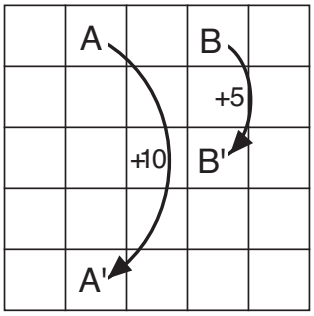


图 1: Grid world.

西, 北和南. 每个动作让智能体朝对应的方向移动一格, 如果智能体因此而出界, 我们保持其位置不动, 并且得到一个 -1 的奖励. 当智能体在 A 点时, 所有的动作都会让智能体转移到 A', 且得到一个 $+10$ 的奖励. 类似地, 当智能体在 B 点时, 所有动作都会让它转移到 B', 且得到一个 $+5$ 的奖励. 其他动作的奖励为 0. 这是一个持续型的 MDP 模型 (没有终止状态), 我们设定折扣系数为 $\gamma = 0.9$.

2 最优价值函数和最优策略

为了得到最优价值函数和最优策略, 根据 Bellman 方程

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)(r + \gamma v_*(s')) \tag{1}$$

和

$$q_*(s,a) = \sum_{s',r} p(s', r \mid s, a)(r + \gamma \max_{a'} q_*(s', a')), \tag{2}$$

我们可以将式 (1) 和 (2) 看成关于 $v_*(s)$ 或者 $q_*(s,a)$ 的方程组, 且 (1) 和 (2) 的形式可以让我们使用迭代法求解. 具体来说, 我们通过

$$v_k(s) = \max_a \sum_{s',r} p(s', r \mid s, a)(r + \gamma v_{k-1}(s')) \tag{3}$$

和

$$q_k(s,a) = \sum_{s',r} p(s', r \mid s, a)(r + \gamma \max_{a'} q_{k-1}(s', a')) \tag{4}$$

进行迭代计算. 这里我们取初始值 $v_0(s) = 0$, $q_0(s,a) = 0$, 迭代停止条件为

$$\frac{\|v_k - v_{k-1}\|_\infty}{\max\{1, \|v_{k-1}\|_\infty\}} \le \epsilon, \quad \frac{\|q_k - q_{k-1}\|_\infty}{\max\{1, \|q_{k-1}\|_\infty\}} \le \epsilon,$$

其中

$$\|v\|_\infty = \max_s\{|v(s)|\}, \quad \|q\|_\infty = \max_{s,a}\{|q(s,a)|\},$$

ϵ 为我们需要的容差.

设定 $\epsilon = 1e^{-4}$ 时, 通过 (3) 和 (4) 得到的最优价值函数和最优策略 (最优策略是通过计算得到的动作-价值函数 $\tilde{q}_*$ 选择的) 分别见图 2 和图 3. 多个箭头的状态表示在此状态下, 任何一个箭头所

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

图 2: Grid World 的最优价值函数.

代表的动作 (或者它们的混合) 都是最优的.

在形如 (3), (4) 的迭代中, 我们是通过遍历状态 s 或者状态-动作 (s,a) 来进行更新的, 我们称这样的一次遍历为一个 sweep. 为了得到图 2 和图 3 的结果, 对于 v 的迭代, 我们需要进行的总的 sweep 个数为 65, 换言之, 我们需要进行 $65 * 25 = 1625$ 次更新. 而对于 q 的迭代, 我们需要进行 65 个 sweep, 或者说, $65 * 25 * 4 = 6500$ 次更新. 由于整个 MDP 对我们来说是已知的, 这里的收敛是非常快的.

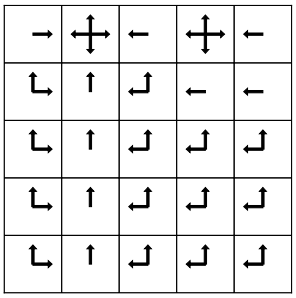


图 3: Grid World 的最优策略.

3 强化学习方法在 Grid World 上的表现

3.1 SARSA, Q-learning, E-SARSA

我们采用 3 种强化学习方法对 Grid World 进行测试: SARSA, Q-learning 和期望 SARSA(E-SARSA), 其中动作的产生由 ε-greedy 给出. 值得注意的是, 我们的任务是持续型的, 没有终止状态. 如果我们从一个初始点开始 (这个初始点可以是任意的), 我们会在 grid world 里不断地进行游走直到达到最大游走次数或者收敛到最优策略, 这与第 2 节中计算最优价值函数和最优动作-价值函数时我们进行的遍历更新 (sweep) 不太相同的. 我们也将看到, 这种特性使得我们的学习变得非常缓慢. 从直观上来说, 我们会倾向于少访问价值相对较低的状态 (可能是图 1 中靠近边界的点) 或者其他不太可能访问到的状态 (例如, 一个状态 S_1 可能本身价值很高, 但是有比 S_1 更好的状态 S_2, 这导致我们更倾向于选 S_2 而非 S_1), 使得它们的更新次数很低, 这种更新次数的不平衡性会影响收敛速度.

我们考虑这样的问题: 收敛到最优策略需要多久? 同第 2 节一样, 我们还是采用更新次数来衡量收敛的快慢, 在 $\varepsilon = 0.1$ 的设定下, 我们取学习步长 $\alpha = 0.1$ 进行测试, 如果当前的策略是最优策略时, 迭代停止. 我们取相同的随机种子, 因此 3 种方法的初始点都是 $(4, 5)$.[1] 3 种方法最终收敛到的最优策略 (最优策略由对计算得到的 $\tilde{q}$ 用 greedy 得到) 由图 4 给出, 可以看到, 此时的策略只是最优策略的一种选择, 这说明这时对价值函数的估计是不那么准确的. 表 1 给出了 3 种方法的更新次数, 可以看到, 与之前对 q 的迭代计算 (4) 相比, 更新次数是显著增加的. 我们也画出此时 3 种方

SARSA	Q-learning	E-SARSA
3574831	2199784	1513973

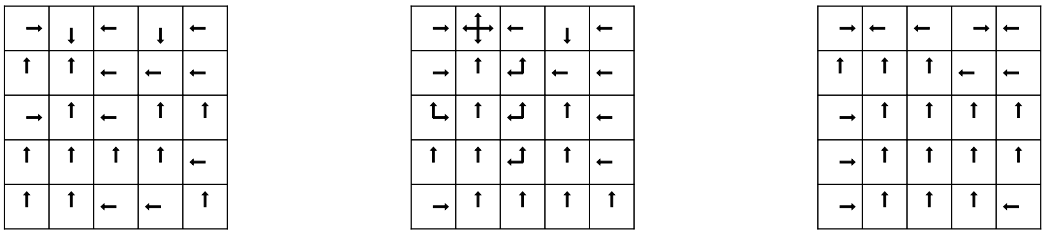
表 1: 3 种方法收敛到最优策略时候的更新次数.

法对应的价值函数, 它由公式

$$\tilde{v}(s) = \max_a \tilde{q}(s, a)$$

给出, 请见图 5. 需要注意的是, 在 on-policy 的方法 (SARSA 和 E-SARSA) 中, 由于我们是按照 ε-greedy 方法生成动作, 我们计算得到的值函数 $\tilde{v}$ 不会是最优策略的值函数 v_*, 但我们仍然可以期待 $\tilde{v}$ 和 v_* 是非常接近的. 对比图 2, 正如我们前面所说, 3 种方法对于某些状态的价值的估计是不那

[1] 在这篇报告中, 坐标是从 1 开始的, 而在我们的代码中, 坐标是从 0 开始的.



(a) SARSA. (b) Q-learning. (c) E-SARSA.

图 4: 3 种方法收敛到的最优策略.

20.3	22.5	15.5	17.3	4.2
18.1	20.3	17.8	16.2	14.2
16.2	17.5	15.9	13.8	11.8
14.6	16.1	14.4	12.5	10.5
12.7	14.2	12.8	11.1	4.4

17.9	24.4	22.0	19.4	16.7
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	12.9
14.4	16.0	14.4	13.0	10.7

20.1	22.3	20.1	17.4	15.4
17.2	20.1	17.8	15.7	13.9
15.7	17.8	15.8	14.0	12.3
13.9	15.7	14.0	12.4	10.8
12.3	13.9	12.4	10.8	7.7

(a) SARSA. (b) Q-learning. (c) E-SARSA.

图 5: 3 种方法收敛到最优策略时对应的价值函数.

么准确的, 比如右下角的点 $(5, 1)$.

我们还可以给出收敛到最优策略时每个状态的更新次数, 请见图 6. 正如我们在本节开头所说的那样, 更新次数是不均衡的. 例如, 在 SARSA 中, 更新次数最多的状态是 $(2, 1)$, 它被更新了 677501 次, 但更新次数最少的状态 $(5, 5)$ 只被更新了 7 次. 这种不平衡性也出现在了 Q-learning 和 E-SARSA 中. 对照图 2 所示的最优价值函数, 在图 6 中, 右下角的点价值相对较低, 因此被访问的次数比较少, 从而被更新的次数也比较少. 我们还注意到, 有些处于边界的状态, 即使它们的价值可能很大, 比如左上角的点 $(1, 5)$, 我们仍然可能很少访问它, 这是因为根据图 3 中的最优策略, 我们的策略几乎总是让智能体选择接近 A, 此后它经历转移到达 A' (也就是被更新的比较多的 (2.1)), 这个时候, 比起选择左边 (这会带来出界的风险), 智能体更愿意选择右边, 这样, 智能体几乎没什么可以接近左上角的点机会. 在 Q-learning 和 E-SARSA 中, 智能体学到的策略甚至是在 $1, 1$ 这个点向右走!

这种更新的不平衡性也会带来收敛的不平衡性, 从而降低我们的收敛速度. 最极端的情况下, 更

22937	632018	746	2523	7
47055	660226	30319	2132	164
40612	658577	36933	4344	121
31146	660686	28075	861	45
18852	677501	18420	506	25

292	323158	46591	98072	68
10550	306311	55088	103965	2783
16707	314733	48818	110033	3722
8703	312182	39507	4979	268
9583	349808	32853	971	39

236	16086	32024	421206	488
243	8083	33776	431702	20526
262	8625	23055	446272	12634
461	17047	9983	11784	629
464	17378	664	320	25

(a) SARSA. (b) Q-learning. (c) E-SARSA.

图 6: 3 种方法收敛到最优策略时每个状态的更新次数.

新次数比较多的状态的 q 可能早已收敛到最优值, 但更新次数比较少的状态的 q 还没有收敛, 但此后我们还是会更多地访问那些 q 值已经收敛到最优的状态, 这显然是不合理的. 我们还可以画出 3 种方法更新 100 万次后的价值函数, 和前面一样, 3 种方法的初始点都是 $(4,5)$. 计算得到的价值函数如图 7 所示. 与图 2 给出的最优价值函数相比, SARSA 和 Q-learning 在某些点 (比如 A 附近的

20.2	22.4	15.5	16.9	4.2
18.0	20.1	18.1	15.3	13.3
15.7	17.9	15.6	13.9	9.4
14.2	15.7	14.2	12.4	5.1
12.7	13.9	12.7	11.2	0.4

(a) SARSA.

17.8	24.4	22.0	19.4	16.7
19.8	22.0	19.8	17.8	15.9
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	12.5
14.4	16.0	14.4	13.0	10.0

(b) Q-learning.

10.0	17.0	15.3	17.0	15.3
3.6	12.1	13.6	15.3	13.6
5.3	10.8	12.1	13.6	12.1
5.6	9.5	10.8	12.0	10.8
3.6	8.2	9.3	10.6	6.5

(c) E-SARSA.

图 7: 3 种方法更新 100 万步后的值函数.

点) 上计算出的最优价值已经相当准确, 而对于靠近边界的点的值函数的估计误差是比较大的, 这在 SARSA 上体现的尤为明显, 特别是对于右下角的点 $(5,1)$ 和右上角的点 $(5,5)$. 而对于 E-SARSA, 此时得到的值函数与实际还差距很大. 不过注意到在表 1 中, E-SARSA 只花了近 150 万步, 这说明在后续的步骤中每个点都能相对很快地收敛.

我们再给出前 100 万次更新中的实时策略占最优策略的动作比例. 我们每隔 10000 步记录在所有状态当在前 q 值下策略 (根据 q 按 greddy 选取) 的动作占最优策略动作的比例. 见图 8. 可以看到, 3 种方法一开始都能够很快地学习, 然而到了一定程度后, 它们的学习速度都会下降, 且总的来说, SARSA 相比其他方法效果相对更差一些.

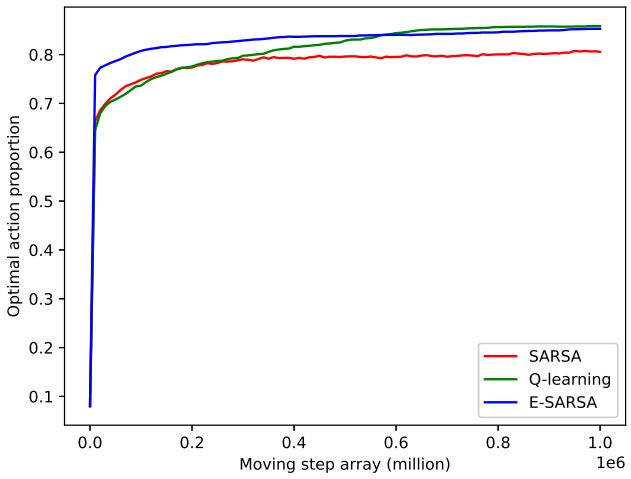
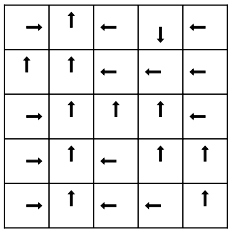
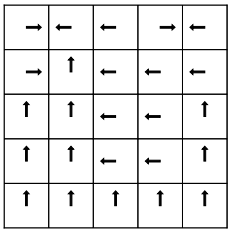


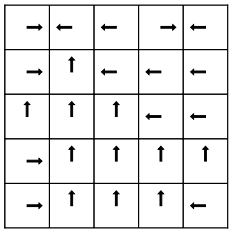
图 8: 前 100 万次更新中的实时策略占最优策略的动作比例.



(a) SARSA.



(b) Q-learning.



(c) E-SARSA.

图 9: $k = 10$ 时 3 种方法收敛到的最优策略.

1500	15976	1962	9745	595
1025	15299	2589	10513	1020
1073	17551	3672	12960	1405
1045	17045	1138	2180	1872
1624	18131	1118	694	1268

(a) SARSA.

568	11237	527	12578	546
2835	12499	1401	14020	2635
2233	11000	2291	14174	1934
1522	10686	2559	2199	1134
770	10998	1499	599	566

(b) Q-learning.

748	18788	1318	12913	743
2362	20655	4527	13324	1085
1273	18904	8335	18175	1970
1215	18124	2502	2658	858
1200	19393	1249	1558	783

(c) E-SARSA.

图 10: $k = 10$ 时 3 种方法收敛到最优策略时每个状态的更新次数.

3.2 在 SARSA, Q-learning, E-SARSA 中引入遍历的平衡性

为了克服第 3.1 节中提到的问题, 加快收敛, 我们还是希望能够在 grid world 上采用遍历性更新, 或者至少能够增加对不常访问状态的更新. 我们这里的想法类似于从 Monte Carlo 方法 (MC), 和 1 步时序差分 (1-step TD) 得到 n 步时序差分 (n-step TD). 引入最大游走次数 k, 在达到最大游走次数时, 我们强制停止并初始化起点重新开始. 由于我们的任务是持续型的, 我们可以将在 grid world 里不断进行游走看作是 $k = \infty$ 的情况, 这也是在第 3.1 中我们遇到的情况. 而第 2 中的迭代 4 可以看成是 $k = 1$ 且初始化起点采用正好遍历方格点的情况 (当然, 在那里我们的更新是根据已知的 MDP 模型而非采样进行的).

我们用这种新的方法进行 SARSA, Q-learning 和 E-SARSA. 我们采用随机初始化起点, 且取 $k = 10$, 它们得到的最优策略和对应的更新次数由图和表 2 给出. 可以看到, 与表 6 相比, 此时的更新次数已经显著下降了 (至少下降了一个量级). 同样地, 我们也给出这种情况下每个状态的更新次

SARSA	Q-learning	E-SARSA
143000	123010	174660

表 2: $k = 10$ 时 3 种方法收敛到最优策略时的更新次数.

数, 结果见图 10. 可以看到, 相比于图 6 中的更新, 在这里我们的更新要更加 “平均”, 从而能够加速收敛.

我们探究 k 的取值与收敛速度的关系. 图 11 展示了不同取值的 k 对更新次数的影响, 我们采用做 100 次实验取平均的方法. 可以看到, 更新次数大致是随着 k 的增加而增加的, 因此, 在第 3.1 中使用的方法可能是最慢的.

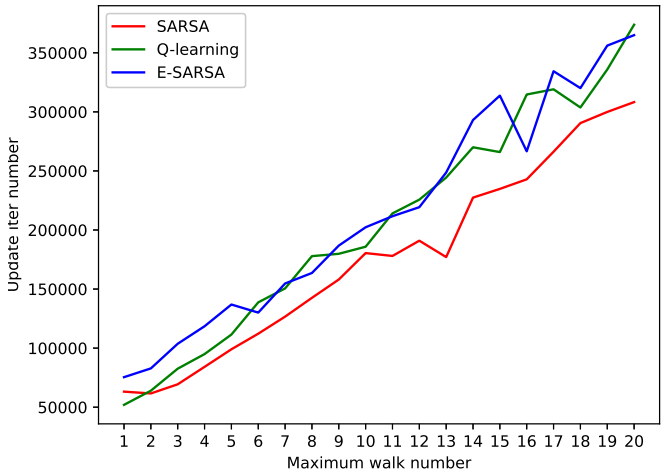


图 11: k 取不同值时的更新次数.

最后, 我们指出, 本节中提出的方法对于问题本身有一定要求. 在 grid world 里, 每一个状态都可以作为起点, 因此我们这里是从起点重新开始. 如果任务本身只有很少的起点或甚至只有一个起点, 我们也可以考虑终止后从任何一个状态重新开始. 此外, 在 grid world 里, 无论 k 取何值, 我们都能保证每一个状态被无限次访问到. 但是对于一些 “深度” 较大的问题 (在 grid world 中 “深度” 其实为 0, 因为每一个状态都可以作为起点, 它们离起点的最大步数为 0), 我们必须要求 k 超过一定值才能满足所有状态都被访问. 如果 k 取的过小, 最坏的情况下, 我们甚至只会在很小的一部分状态里更新, 这显然不会对我们的收敛有任何帮助. 对于这种问题, 我们可以考虑将 k 取的稍微大一些, 也可以考虑终止后从任何一个状态重新开始.