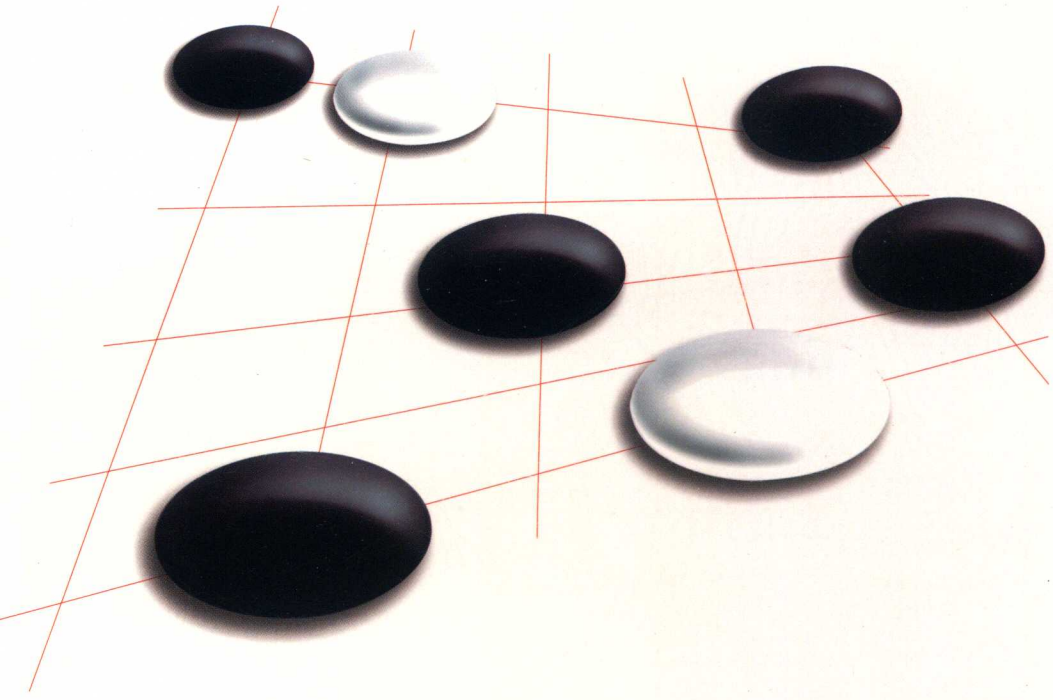
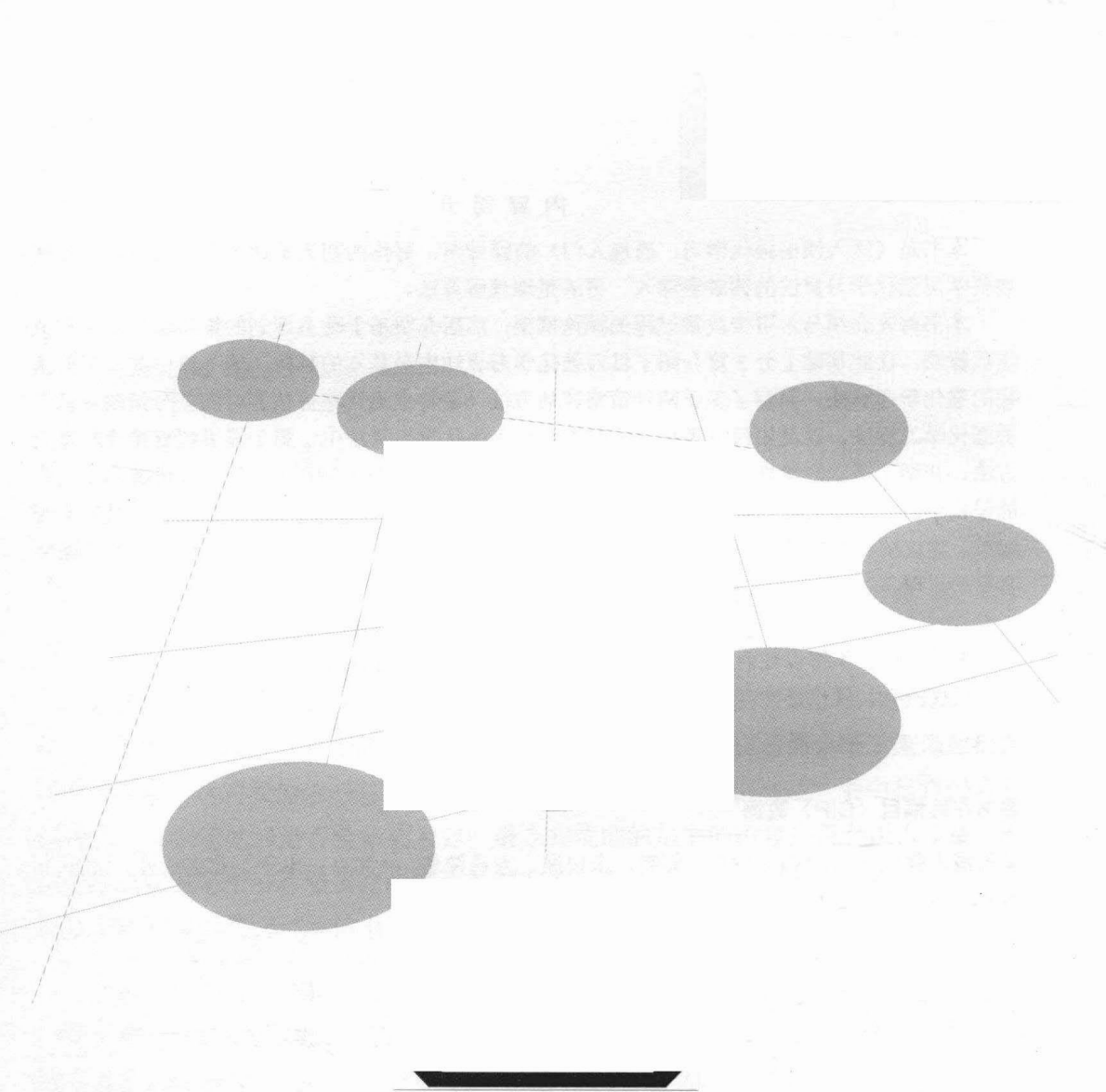
Broadview®
www.broadview.com.cn



深入浅出强化学习
编程实战

郭宪　宋俊潇　方勇纯　著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn



深入浅出强化学习

编程实战

郭　宪　宋俊潇　方勇纯　著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书是《深入浅出强化学习：原理入门》的姊妹篇，写作的初衷是通过编程实例帮助那些想要学习强化学习算法的读者更深入、更清楚地理解算法。

本书首先介绍马尔可夫决策过程的理论框架，然后介绍基于动态规划的策略迭代算法和值迭代算法，在此基础上分 3 篇介绍了目前强化学习算法中最基本的算法。第 1 篇讲解基于值函数的强化学习算法，介绍了基于两种策略评估方法（蒙特卡洛策略评估和时间差分策略评估）的强化学习算法，以及如何将函数逼近的方法引入强化学习算法中。第 2 篇讲解直接策略搜索方法，介绍了基本的策略梯度方法、AC 方法、PPO 方法和 DDPG 算法。第 3 篇讲解基于模型的强化学习方法，介绍了基于 MPC 的方法、AlphaZero 算法基本原理及在五子棋上的具体实现细节。建议读者根据书中的代码亲自动手编程，并修改程序中的超参数，根据运行结果不断体会算法原理。

图书在版编目（CIP）数据

深入浅出强化学习. 编程实战 / 郭宪，宋俊潇，方勇纯著. —北京：电子工业出版社，2020.3
ISBN 978-7-121-36746-5

Ⅰ. ①深… Ⅱ. ①郭… ②宋… ③方… Ⅲ. ①人工智能－程序设计 Ⅳ. ①TP18

中国版本图书馆 CIP 数据核字(2019)第 111720 号

责任编辑：刘　皎
印　　刷：三河市君旺印务有限公司
装　　订：三河市君旺印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：720×1000　1/16　印张：17　字数：354 千字
版　　次：2020 年 3 月第 1 版
印　　次：2020 年 3 月第 1 次印刷
定　　价：89.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前　言

每次写书的前言时，我总会感到惴惴不安。因为写前言的时候，书往往已经定稿了，不能再修改内容，书中的每一个字、每一句话、每一段表述都会毫无掩饰地袒露在读者面前。这时我的内心总是惶恐：书中的内容是否已经讲清楚？书中是否仍有很多漏洞？书中的见解是否会贻笑大方？最令我忧虑的是书的内容是否会误人子弟。相信这些担忧是每个心存敬畏之心的作者都有的。直到自己写了两本书，才真正明白那句从小就知道的名言“尽信书不如无书”！

不过，开卷有益，我们还是要多读书、读好书，取之精华，去之糟粕。一本书或一篇文章存在的价值就在于它的精华，我希望这本书还有那么一点点的精华，能帮助到不同层次的读者。

这本书的姊妹篇是《深入浅出强化学习：原理入门》，该书绝大部分的笔墨都用在描述算法的原理上，至于算法的实现，讲得并不多。《深入浅出强化学习：原理入门》出版后，我意识到学习强化学习算法如同学游泳，只知道理论而不下水去游，就永远学不会；同理，只懂原理而没有进行编程训练，永远学不会强化学习算法。出于这样的动机，我为《深入浅出强化学习：原理入门》一书编写了配套程序。在程序的编写过程中，我参考了很多网上的资源，如《莫烦 Python》、GitHub 上的很多源代码等。同时还邀请了知乎上的好友——网名为“一缕阳光”的宋俊潇共同完成这本以编程为主题的书。在这里非常感谢《莫烦 Python》，感谢那些将代码开源的作者们。

本书写作的初衷是通过编程实例帮助那些想要学习强化学习算法的读者更深入、更清楚地理解算法。本书的篇章结构与《深入浅出强化学习：原理入门》一书的篇章

结构大体相同，首先介绍马尔可夫决策过程的理论框架，然后介绍基于动态规划的策略迭代算法和值迭代算法，在此基础上介绍了目前强化学习算法中最基本的算法。第1篇讲解基于值函数的强化学习算法，该篇介绍了基于两种策略评估方法（蒙特卡洛策略评估和时间差分策略评估）的强化学习算法，进而介绍了如何将函数逼近的方法引入强化学习算法中。第2篇讲解直接策略搜索的方法，该篇介绍了最基本的策略梯度方法、AC方法、PPO方法和DDPG算法。第3篇讲解基于模型的强化学习方法，该篇介绍了基于MPC的方法、AlphaZero算法基本原理及在五子棋上的具体实现细节。建议读者根据书中的代码亲自动手编程，并修改程序中的超参数，根据运行结果不断体会算法原理。笔者在为本书编写代码的过程中也受益匪浅。

本书第1篇中基于值函数方法的代码编程得益于笔者在2018年下半年给研究生开的选修课。很多代码参考了学生的作业，在该课程方面表现突出的有：冯帆、曹丁元、郑昊思等，在此感谢。本书第2篇中直接策略搜索的方法中很多代码则参考了莫烦的Python课程。第3篇中基于模型的强化学习算法的代码则参考了伯克利CS294深度强化学习课程的作业，AlphaZero部分则基于宋俊潇于2018年上半年完成的一个GitHub开源项目。课题组内的朱威、赵铭慧、张学有、姜帆、戚琪和古明阳等人也参与了本书的讨论和校对，没有他们，这本书也不可能完成。总之，这本书参考了很多网络资源，非常感谢互联网的共享精神，感谢那些拥有共享精神的网友，免费、共享，是计算机学科能快速发展的重要原因。

非常感谢方勇纯教授对笔者的教导和帮助，并积极推进本书的进展。感谢编辑刘皎女士默默无闻的辛勤付出，感谢国家自然基金青年基金（61603200）对笔者的支持。

最后，非常感谢我的爱人王凯女士，在历经艰难的十月怀胎后，生了一个健康漂亮的小公主，非常感谢岳父岳母日日夜夜不辞辛苦地呵护着宝宝健康成长。这本书献给最可爱最聪明最美丽的女儿汐汐。

郭　宪

2019年11月

目　录

Part Zero

第0篇

先导篇

本篇名为先导篇，目的是介绍强化学习算法的基本理论框架。第1章用很简单的例子让读者了解强化学习算法的流程，以及探索和利用平衡的问题。第2章介绍了强化学习算法的基本理论框架——马尔可夫决策过程。

1 一个极其简单的强化学习实例

强化学习算法最基本的理论框架是马尔可夫决策过程，在正式进入对马尔可夫决策过程的讲解和编程之前，我们先介绍一个极其简单的例子：多臂赌博机问题。对于该例子，我们利用强化学习的思想进行解决。在解决的过程中，读者们可以体会到强化学习算法中最核心的思想，如交互学习、探索和利用平衡等概念。

本章 1.1 节给出多臂赌博机的描述和基本算法，1.2 节给出完整的代码实现，1.3 节给出知识点总结和笔者的一些个人体会。

1.1 多臂赌博机

多臂赌博机（Multi-Armed Bandit）是指一台拥有 K 个臂的机器（如图 1.1 所示，图中的多臂赌博机有 3 个臂，即 K=3），玩家每次可以摇动这 K 个臂中的 1 个臂，摇动摇臂后，多臂赌博机会吐出数量不等的金币，吐出金币的数量服从一定的概率分布，而且该概率分布根据摇臂的不同而变化。也就是说，摇动不同的摇臂，多臂赌博机就会吐出数量服从不同概率分布的金币。用更通俗的话来说，有时摇臂吐出的金币多，有时摇臂吐出的金币少。

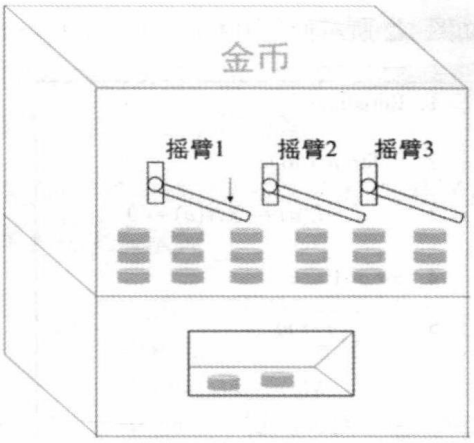


图 1.1　多臂赌博机示意图

多臂赌博机的问题是：假设玩家共有 N 次摇动摇臂的机会，每次如何选择摇动哪个臂以使得 N 轮后得到的金币最多？

对于这个问题，如果你提前知道每个臂对应吐出多少金币，那么每次都摇动那个吐金币最多的臂就可以了。但是，问题是你并不知道摇动哪个臂能获得最多金币。这个时候，你该采用什么样的策略去玩这 N 次游戏，从而得到最多的金币呢？

1.1.1　$\varepsilon-$greedy 策略

一个很直观的想法是：既然不知道哪个臂吐的金币最多，那么可以先对每个臂都尝试几次（如都尝试 10 次），找出哪个臂吐的金币最多，然后一直摇动它。

其实这个最简单、最朴素的想法已经蕴含了算法学习最基本的两个过程：采集数据和学习。首先，对每个臂进行尝试就是采集数据：每次尝试时，多臂赌博机会吐出不同数量的金币，这些不同数量的金币就是数据；其次，学习就是利用这些数据知道哪个臂会吐出最多的金币。一个最简单的思想是计算每个臂的平均吐钱数量。然后，我们一直摇那个吐钱最多的臂。

我们可以将这个算法用更形式化的代数来表示。用 s 表示当前多臂赌博机，用 A 表示可以选择的动作，即 $A=\{1,2,3\}$，其中 a=1 表示摇动第 1 个臂，a=2 表示摇动第 2 个臂，a=3 表示摇动第 3 个臂。用回报 r 表示摇动赌博机的摇臂后所获得的金币的数目。用 $Q(a)$ 表示摇动动作 a 所获得的金币的平均回报，则 $Q(a)=\dfrac{1}{n}\sum_{i=1}^{n}r_i$，其中 n 为摇动动作 a 的总次数。$R(a)$ 为摇动动作 a 的总回报。有了这些字母，上面最简单、最朴

素的算法伪代码的表述如图 1.2 所示。

1. Initialize：

2. for a=1 to k:

3. $R(a) \leftarrow 0, N(a) \leftarrow 0$

4. for a=1 to k:

5. for i=1 to n_a :

6. $R(a) \leftarrow R(a) + r_i(a)$

7. for $t = n_1 + \cdots + n_k, \cdots, N$

8. a=argmax $R(a)$

9. $R(a) = R(a) + r_a$

10. return $R(a)$

图 1.2 最简单的算法伪代码

在本书中会出现很多伪代码，它们可以把编程的思路描述得非常清楚。所以，读者阅读本书时请不要跳过伪代码。

伪代码的解释：

第 1~3 行：初始化每个动作的总回报 $R(a)$，以及摇动该动作的次数 $N(a)$。

第 4~6 行：每个臂都尝试n_a次，计算每个摇臂总的金币数。

第 7~9 行：算出使得总回报最大的那个臂，一直摇动它。

这是一个简单而朴素的想法，但并不是一个好的算法。原因如下。

第 1 个缺点是：我们不应该以总回报最大为目的来选择当前摇哪个臂，而是应该选择使得当前平均回报最大的臂。因为在后面的摇动过程中，经过n_a次摇动后使总和最大的那个臂的平均回报可能会变小，而平均回报才是能真正反映臂好坏的量。所以伪代码的第 8 行应该比较当前每个臂的平均回报，摇动使平均回报最大的臂。

第 2 个缺点是：我们不应该只摇动当前平均回报最大的臂，因为它不一定是最好的那个臂。所以，我们除了要关注当前平均回报最大的那个臂，还要保留一定的概率去摇动其他的臂，以便发现更好的臂。

以上两点分别对应着强化学习算法中最重要的概念：利用策略和探索策略平衡。

其中“利用”就是所谓的 exploitation，是利用当前的数据总结得到的最好的策略，采用该策略，我们可以得到比较高的回报。而“探索”就是所谓的 exploration，该策略能够帮助我们找到更好的臂，甚至找到最优的臂。

强化学习算法在训练过程中所用到的策略是平衡了利用和探索的策略，最常见的是 ε – greedy 策略。该策略用公式表示为

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{概率为} 1-\varepsilon \\ \text{随机动作} & \text{概率为} \varepsilon \end{cases} \tag{1.1}$$

该策略的意思是，在每次选择摇动哪个臂的时候，应该以 $1-\varepsilon$ 的概率去摇动当前均值最大的那个臂，以 ε 的概率在所有的动作中均匀随机地选动作。这样做的目的是在有限的次数中得到尽可能多的回报，同时不失去找到最好的臂的机会。ε – greedy 算法的伪代码如图 1.3 所示。

1. Initialize:
2. $R(a) \leftarrow 0$
3. for a=1 to k:
4. $Q(a) \leftarrow 0, N(a) \leftarrow 0$
5. for t=1 to N:
6. $a \leftarrow \begin{cases} \arg\max_a Q(a) & \text{概率为} 1-\varepsilon \\ \text{随机动作} & \text{概率为} \varepsilon \end{cases}$
7. $N(a) \leftarrow N(a)+1$
8. $Q(a) \leftarrow Q(a)+\frac{1}{N(a)}[r(a)-Q(a)]$
9. $R(a)=R(a)+r(a)$
10. retrun $R(a)$

图 1.3　ε – greedy 算法的伪代码

伪代码解释：

第 1~4 行：初始化总回报 R，初始化每个动作的平均回报 $Q(a)$，每个动作的次数 $N(a)$。

第 6 行：在每次摇臂之前，利用 ε – greedy 策略选择要摇动的臂 a。

第 7 行：动作 a 的次数 $N(a)$加 1。

第 8 行：根据动作 a 和环境返回的回报$r(a)$，更新动作 a 的平均回报。

第 9 行：计算总的收益。

第 10 行：玩家尝试 N 次后，返回总的收益。

总结：

（1）$\varepsilon-$greedy 策略是探索和利用平衡的策略，这是强化学习算法能够学习到最优策略的关键所在。

（2）强化学习算法在训练的过程中采用不断更新的策略 $\varepsilon-$greedy 与环境（多臂赌博机）进行交互。所产生的回报数据$r(a)$是交互数据，强化学习算法从交互数据中学习。

在多臂赌博机问题中，我们平衡利用和探索的策略还有玻尔兹曼（Boltzmann）策略和 UCB 策略。我们先介绍玻尔兹曼策略。

1.1.2 玻尔兹曼策略

前面介绍的 $\varepsilon-$greedy 策略是平衡利用与探索的策略，其中利用部分的概率，即值函数最大的动作对应的概率为$1-\varepsilon+\frac{\varepsilon}{|A|}$，而其他的动作被选择的概率为$\frac{\varepsilon}{|A|}$。从直观上看，$\varepsilon-$greedy 策略给对应值函数最大的那个动作一个比较大的概率，即$1-\varepsilon+\frac{\varepsilon}{|A|}$，而其他的动作，不管对应的值函数的大小如何，被采样的概率都是相等的，即$\frac{\varepsilon}{|A|}$。这种概率的分配方式有些不合理。按理说，非贪婪的动作也有好坏之分，那些对应值函数大的动作应该比那些对应值函数小的动作被采样的概率大，而 $\varepsilon-$greedy 策略并没有对非贪婪策略进行区分，只是很僵硬地将它们的概率进行统一处理。玻尔兹曼策略则根据对应的值函数对动作采样的概率进行了软（Soft）处理。具体的玻尔兹曼策略表示为

$$p(a_i)=\frac{\exp(\frac{Q(a_i)}{\tau})}{\sum_{j=1}^{K}\exp(\frac{Q(a_j)}{\tau})} \tag{1.2}$$

在式（1.2）中τ为温度调节参数，可用来调节探索和利用的比例。τ越小，玻尔兹曼策略越接近贪婪策略，利用所占的比例越大，探索越少；τ越大，玻尔兹曼策略越接近均匀策略，探索就越多。利用玻尔兹曼策略进行学习的伪代码与图 1.3 类似，不同的是用式（1.2）代替了第 6 行的$\varepsilon-$greedy 策略。

1.1.3　UCB 策略

UCB 的全称是 Upper Confidence Bound(置信上界)，在统计学中常常用置信区间来表示不确定性。在这里，我们用置信区间来表示探索，具体解释请参看本书后面的章节。在这里，我们只给出 UCB 策略的公式：

$$A_t = \arg\max_a [Q(a) + c\sqrt{\frac{\ln t}{N(a)}}] \tag{1.3}$$

其中t为当前摇臂动作的总次数，$N(a)$为动作a的总次数。

如图 1.4 所示为分别采用 3 种学习策略时，总回报与摇动次数的关系。从图中我们看出 UCB 策略回报最高，玻尔兹曼策略次之，$\varepsilon-$greedy 策略回报最低。然而，$\varepsilon-$greedy 策略在 3 种策略中形式最简单、最通用，可广泛用于各种任务的学习和探索训练中。

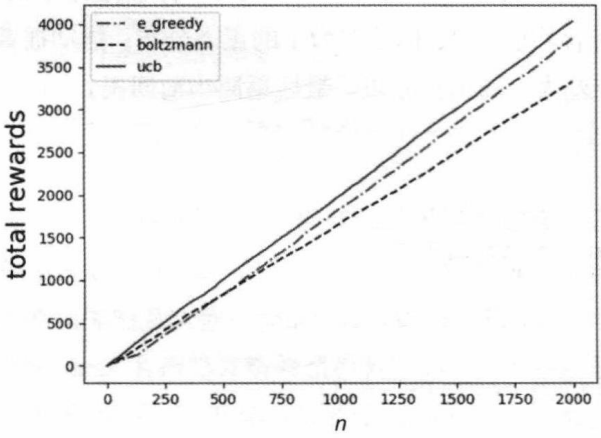


图 1.4　3 种策略的总回报

1.2　多臂赌博机代码实现

本节基于 Python 实现上一节介绍的 3 种学习策略。首先，我们创建一个 KB_Game

类，该类包括以下几个属性：q（每个臂的平均回报，在这里，我们假设臂的数目为3个，初始值都为 0.0）、action_counts（摇动每个臂的次数，初始值为 0）、current_cumulative_rewards（当前累积回报总和，初始值为 0.0）、actions（动作空间，我们用 1、2、3 分别表示 3 个不同的摇臂）、counts（玩家玩游戏的次数）、counts_history（玩家玩游戏的次数记录）、cumulative_rewards_history（累积回报的记录）、a（玩家当前动作，初始值可为动作空间中的任意动作，这里去摇动第一个臂），reward（当前回报，初始值为 0）。

```
class KB_Game:
def __init__(self, *args, **kwargs):
    self.q = np.array([0.0, 0.0, 0.0])
    self.action_counts = np.array([0,0,0])
    self.current_cumulative_rewards = 0.0
    self.actions = [1, 2, 3]
    self.counts = 0
    self.counts_history = []
    self.cumulative_rewards_history=[]
    self.a = 1
    self.reward = 0
```

在类 KB_Game 中定义方法 step()，用于模拟多臂赌博机如何给出回报。该方法的输入为动作，输出为回报。我们用正态分布来模拟玩家在每次摇动摇臂后得到的回报。其中，假设摇动摇臂 1，得到的回报符合均值为 1、标准差为 1 的正态分布；摇动摇臂 2，得到的回报符合均值为 2、标准差为 1 的正态分布；摇动摇臂 3，得到的回报符合均值为 1.5、标准差为 1 的正态分布。最后返回当前回报。

```
def step(self, a):
r = 0
if a == 1:
    r = np.random.normal(1,1)
if a == 2:
    r = np.random.normal(2,1)
if a == 3:
    r = np.random.normal(1.5,1)
return r
```

上面的 KB_Game 类的 step()方法实际上提供了多臂赌博机的模拟器。接下来，我们要实现的是 3 种选择动作的策略方法 choose_action()。该方法的输入参数为策略类别（policy），在本程序中，有 3 个 policy，分别是“e_greedy”“ucb”和“boltzmann”，对应 1.1 节的 3 个策略；另外还有一个参数字典**kwargs，用于传递相应的策略所对