目 录

第1章 《RL:An Introduction》阅读笔记

先通读全文了解基本脉络，然后针对关节点逐一深入。对阅读过程中的问题进行笔记，把之前的纸质版笔记录入，也是一个对思路进行整理的过程。

1.1 Introduction

In this book we explore a computational approach to learning from interaction. 提出主旨，也指明了与其它机器学习方法的区别和联系。

RL 的基本目标：Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.

RL 的理论建模方法：MDP

RL 与监督学习

RL 与无监督学习

RL 的主要挑战：处理经验利用与未知探索之间的关系。（One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-o between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be e ective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it has already experienced in order to obtain reward, but it also has to explore in order to make better action selections in the future. ）

另外一个挑战：Another key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment.

一些应用和举例（略）

RL 的主要元素Beyond the agent and the environment, one can identify four main subelements of a reinforcement learning system: a policy, a reward signal, a value function, and, optionally, a model of the environment.

A policy de nes the learning agent’s way of behaving at a given time. A reward signal de nes the goal in a reinforcement learning problem. the value of a state is the

total amount of reward an agent can expect to accumulate over the future, starting from that state. A model of the environment. This is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave.

RL的发展历史（有空看看，也许有所启发）

注：能够在名字的结尾加-ing的都表示一下三个方面同时存在却又不能混淆： a class of solution methods that work well on the problem, and the eld that studies this problem and its solution methods.

1.2 (Tabular Solution Methods)Multi-armed Bandits

第一部分引子：本部分所叙述的强化学习的基本方法，但是在简单条件下进行讨论，所谓的简单条件是把状态空间和动作空间限制在一个比较小的有限空间，此时的值函数可以由矩阵或者表格进行表示。这种情况下问题通常有一个明确的最优解。而在下一个part我们要讨论的问题是在连续的无限空间中进行，通常更符合实际情况，但是得到的都是问题的近似解。（approximation）

首先介绍的一个简单例子中只有一个简单的状态。接下来介绍了RL的通用建模方法，MDP。

接下来的三章中介绍了有限马尔可夫决策过程的三个基本解决方法：动态规划(DP)，蒙特卡洛采样(MC)，时间差分(TD)。这其中，动态规划方法是问题的原始解决方案，但是需要模型完全已知。MC方法不需要对模型完全已知，但是需要等到每个模拟过程结束，才能进行状态更新。TD方法不需要模型而且是增量式学习，但是分析起来稍显复杂。

接下来的两章则是在讨论如何将这三种方法结合起来，能够发挥各自的优势。MC 和TD, TD 和DP

1.2.1 Multi-armed Bandits

在强化学习中，我们用来训练的数据，是评价动作的而不是指导动作的。也就是我们得到的反馈是刚刚已经做出的动作好到什么程度，而不是什么动作是正确的动作。evaluative feedback depends entirely on the action taken, whereas instructive feedback is independent of the action taken.

本章刚开始的一个假设是nonassociative. the study here does not involve learning to act in more than one situation.

1.2.2 A k-armed Bandit problem

1.2.3 Aciton Value funtion

$Q_t(a)$的引入。 greedy action 的引入。$A_t = argmax_a Q_t(a)$

1.2.4 The 10-armed Testbed

对比$\epsilon - greedy$ 和 $greedy$ 之间的区别.

1.2.5 Incremental Implementation

$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$

这种增量的思想似曾相识，有空总结一下都什么地方会用到这种增量式的转换。文中也总结了一般形式：$NewEstimate_{,}OldEstimate+StepSizehTarget`OldEstimatei$

在此处给出一个简易的求解Q值算法，核心思想是不断的迭代更新Q并应用$\epsilon - greedy$方法获得策略。

1.2.6 Tracking a nonstationary problen

这里重新提了一下什么是nonstationary, 即指奖励函数所服从的分布随着时间进行变化，give more weight to recent rewards than to long-past rewards.

如前文中对增量式更新$Q_{n+1} = Q_n + \alpha[R_n - Q_n]$ 进行进一步展开得到$Q_{n+1} = (1-\alpha) + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$

1.2.7 opotimistic initial values

Initial action values can also be used as a simple way to encourage exploration.

1.2.8 Upper-Confidence-Bound Action Selection

首先对比了一下greedy 和$\epsilon - greedy$在exploration和exploitation上的区别，提出UCB 能够很好地协调二者的关系，It would be better to select among the non-greedy actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal and the uncertainties in those estimates.

这里是在讨论动作选择问题： $A_t = argmax_a[Q_t(a) + c\sqrt{\frac{lnt}{N_t(a)}}]$ 它是一个确定性策略（相对于ϵ-greedy）,但是倾向于采取目前探索次数较少的动作。以此来平衡探索和利用之间的关系。其中，$N_t(a)$是采取动作a的次数。但是在后来用的比较少。

1.2.9 Gradient Bandit Algorithms

不再依赖action values，而是为每个动作定义了一个$H_t(a)$,叫action preference，但并不了解这个举措的后续价值是什么？这个$H_(t)$与V，Q均无关，只与动作π_t以及回报R_t有关。

1.2.10 Associative Search

associate di erent actions with di erent situations.本节将前面的nonassociate问题扩展到简单的associate问题中来。学术中叫做contextual bandits，If actions are allowed to a ect the next situation as well as the reward, then we have the full reinforcement learning problem.也就是说我们的动作不仅会得到不同的回报，还会对环境产生改变，并且影响到下一个动作。

1.2.11 summary

在本章中，为了平衡经验利用和对未知的探索，提出了几个简单直接的方法，包括$\epsilon - greedy$ UCB 和 Gradient bandit algorithms。他们都有各自不同的侧重点。虽然这几种方法对RL问题有一定的普适性，但是，在复杂的环境中会有更好的方法来平衡二者之间的关系。比如，Bayesian methods assume a known initial distribution over the action values and then update the distribution exactly after each step (assuming that the true action values are stationary).

1.3 Finite Markov Decision Process

MDPs are a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards. Thus MDPs involve delayed reward and the need to tradeff immediate and delayed reward.

MDPs are a mathematically idealized form of the reinforcement learning problem for which precise theoretical statements can be made.

1.3.1 The Agent–Environment Interface

什么是agent ？什么是Environment?二者如何进行信息交互？边界是什么？The learner and decision maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment.

详见Figure 3.1,它们的交互可以用状态，动作，回报来表示。其中，动作

是agent向Env的输出，状态和回报时Env向agent的反馈。如果将时刻离散化，得到的序列： $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots$ Env被建模为 $p(s', r|s, a) = Pr(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$

The general rule we follow is that anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment. In general, actions can be any decisions we want to learn how to make, and the states can be anything we can know that might be useful in making them. The agent–environment boundary represents the limit of the agent's absolute control, not of its knowledge. The agent–environment boundary can be located at different places for different purposes. 所有不能主观改变的都可以称作是环境，但是环境并不一定是完全未知的。 A-E的边界限定了agent是否完全可控，并不限定agent的认知边界。

1.3.2 Goals and Rewards

That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

1.3.3 Returns and Episodes

agent's goal is to maximize the cumulative reward it receives in the long run. 所谓的cumulative reward 是指 returns；而long run 的一种情况便是 episodes。具体的定义：

$G_t = R_1 + R_2 + R_3 + \ldots$

$G_t = R_1 + R_2 + R_3 + \ldots + R_T$

$G_t = R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

$G_t = R_t + \gamma G_{t+1}$

$\gamma -> 0$ 是我们常说的短视，$\gamma -> \infty$是远视。

1.3.4 Unified Notation for Episodic and Continuing Tasks

统一对episodic 和 Continuing 的定义：(episodic tasks)：the agent–environment interaction naturally breaks down into a sequence of separate episodes. Rather than one long sequence of time steps, we need to consider a series of episodes, each of which consists of a finite sequence of time steps.

主要考虑有限和无穷的问题： $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ T可以是∞

1.3.5 Policies and Value Functions

几乎所有的RL问题都要讨论值函数的估计： estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state). 在已知的策略π下，某个状态s的值函数为$v_\pi(s)$，is the expected return when starting in s and following π thereafter.

$v_\pi(s) = E[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$

另一种是动作状态值函数：

$q_\pi(s,a) = E[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$

关于策略： a policy is a mapping from states to probabilities of selecting each possible action.

贝尔曼方程：

$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$

1.3.6 Optimal Policies and Optimal Value Functions

优化的目标是： finding a policy that achieves a lot of reward over the long run. 这里所谓的最优策略是对于所有状态来说该策略能够获得最大的长期收益。
$v_*(s) = max_\pi v_\pi(s), wheres \in S$

$q_*(s,a) = max_\pi q_\pi(s,a), wheres \in Sanda \in A$

二者之间关系： $q_*(s,a) = E[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]$

对应最优策略的贝尔曼方程 $v_*(s) = max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_*(s')]$ $q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma max_{a'} q_*(s',a')]$ 这是n个方程组成的方程组，n个未知数，理论上只要p存在就有可能有解。下一章就介绍了如果p已知要如何求解MDP问题，再之后的章节会介绍更现实的情况，就是p不知道，我们要怎样去估计MDP问题。

1.3.7 Optimality and Approximation

讨论最优化策略的求得，以及估计最优策略的必要性。

1.3.8 summary

几个基础概念：Reinforcement learning is about learning from interaction how to behave in order to achieve a goal. The reinforcement learning agent and its environment interact over a sequence of discrete time steps.

The specification of their interface defines a particular task: the actions are the choices made by the agent;

the states are the basis for making the choices;

and the rewards are the basis for evaluating the choices.

Everything inside the agent is completely known and controllable by the agent;

everything outside is incompletely controllable but may or may not be completely known.

A policy is a stochastic rule by which the agent selects actions as a function of states.

The agent’s objective is to maximize the amount of reward it receives over time.

When the reinforcement learning setup described above is formulated with well defined transition probabilities it constitutes a Markov decision process (MDP). A finite MDP is an MDP with finite state, action, and (as we formulate it here) reward sets. Much of the current theory of reinforcement learning is restricted to nite MDPs, but the methods and ideas apply more generally.

The return is the function of future rewards that the agent seeks to maximize .

A policy’s value functions assign to each state, or state–action pair, the expected return from that state, or state–action pair, given that the agent uses the policy.

四个贝尔曼方程。

1.4 Danamic Programming

如何处理环境模型完全已知的MDP问题。核心是用值函数来组织和规划最优策略，那么，本章就讨论如何用DP的方法来求值函数。基本思路是，从贝尔曼方程出发，用迭代更新的方式对$v_*(s)$进行求解。

1.4.1 Policy Evaluation (Prediction)

策略评估，给定π，评估每个状态的$V_\pi(s)$.

$V(s) <- \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

这里所谓的迭代策略更新：用的是下一个状态的期望值而不是采样值，原文：All the updates done in DP algorithms are called expected updates because they are based on an expectation over all possible next states rather than on a sample next

state. 也就是说即使是在仿真环境中，下一个状态是否是s'也不一定。在迭代过程中，上一个$V_k(s)$整体不变，当一轮迭代完成之后统一更新。

1.4.2 Policy Improvement

策略提升，给定当前的状态值函数与r评价在状态s上根据不同策略π'采取各种动作a所能得到的回报有多少,然后在各种π'中选择一个较优的策略出来。这里不能理解成选择动作a,而是要提升策略π。用较好的代替原来的。 $q_\pi[s,a] = \sum_{s,r} p(s',r|s,a)[r + \gamma V_\pi(s')]$

1.4.3 Policy Iteration

一系列策略评估和策略提升的过程叫做策略迭代。

有一个结论，对于有限的MDP一定能够通过策略迭代的方式找到最优策略。

1.4.4 value iteration

值迭代是针对策略迭代中每次对整个状态集合进行策略评估耗去太多时间的问题提出的。

在值迭代中：

$v_{k+1}(s) = max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$

与策略迭代中

$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$

相比，值迭代中每次迭代用的都是最大的值，而不是期望值（或加权平均值）

1.4.5 Asynchronous Dynamic Programming

异步动态规划

针对前文中的问题是：无论是策略迭代还是值迭代，讨论的基础都是能够遍历整个状态空间。而此处提出的异步动态规划在选取状态时更加随意，原文是这样描述的：These algorithms update the values of states in any order whatsoever, using whatever values of other states happen to be available. The values of some states may be updated several times before the values of others are updated once.那么问题是：为啥可以这样做，如何跳过或者减少一些state的value update，且仍不会改变对策略的正常更新呢？据说会在第八章中进行详细讲解，拭目以待。

这里提出一个有趣的点可以供大家参考，异步算法可以实时与现实情况进行交互。为了求解一个MDP，可以在迭代DP的同时，实际的agent就在现实环境中实现MDP.想法很不错，怎么实现呢？

1.4.6 Generalized Policy Iteration

这里统一了强化学习方法论的一个基本框架：GPI. We use the term generalized policy iteration (GPI) to refer to the general idea of letting policy evaluation and policy improvement processes interact, independent of the granularity and other details of the two processes.

这个框架在书中有两幅图很形象的表示，另外要注意的是，并不一定要一个环节做完再去实现另外一个环节，二者可以同时进行。

1.4.7 Effciency of Dynamic Programming

对大规模问题，not practical!

1.4.8 summary

本章介绍了模型已知时求解MDP的基本方法——动态规划方法。从策略估计，固定当前策略，估计各个状态的值函数；到策略提升，固定当前状体值，改善策略；然后将二者进行结合得到策略迭代方法。为了能节约每次迭代的计算量，又提出了值迭代的方法，该方法中用每次迭代的最大值代替了策略迭代中的期望值。

这里总结了四个基础的值函数，以及其对应的贝尔曼方程：

并将RL中的DP问题求解抽象为GPI:一、假设策略已知，寻求在该策略下与真实value最接近的value function；二、把value function看成已知，提升策略，二者或间歇性或同时进行，直至$V_\pi(s)$不再变动。

最后，异步动态规划又给出了一种不遍历所有状态也能解决问题的可能性。

1.5 Monte Carlo Methods

MC方法中不再假设agent对Env完全已知，只需要：state of agent , actions, rewards from Env. 之所以这样做是因为通常我们得到一个真实环境的回报和一个虚拟环境的采样是比较容易的，但是要得到环境的整体分布是很难的。这里也强调我们用的experience可以是actual也可以是simulated。

本章针对的是episodic task(中文到底要怎么翻译这个词)，即假设每次采用都会到达终止状态。基本方法是averaging complete sample returns，是每个回合更新一次。本章讨论仍然符合GPI框架。

1.5.1 Monte Carlo Prediction

假设策略不变，然后用MC的方法估计状态值函数$V_\pi(s)$. 另外，本节主要讨论first-visit MC, every-visit MC将在C9C12中进行讨论。即只对第一次出现的state进行估计。

估计的基本方法是什么？为啥可行？有啥优势？

基本思想：average the returns observed after visits to that state. As more returns are observed, the average should converge to the expected value.

具体算法见书中算法图：注，这里体现first visit 的地方在“Unless St appears in S0,S1,...,St　1: ”.据说是无偏估计，且标准差是$1/\sqrt{n}$.

主要优势在于，MC对状态的估计是独立的，虽然采样同样是以序列的形式出现，但是 do not bootstrap.这个性质允许我们只估计感兴趣的状态。

1.5.2 Monte Carlo Estimation of Action Values

由于我们是在无模型的基础上讨论MC方法，所以只使用状态值函数不足以进行策略提升，所以本节介绍状态动作值函数$Q_\pi(s,a)$。它仍然属于策略估计的环节。

文中指出，通过采样的方法来估计状态-动作对的值有一点不靠谱的地方，就是有一些状态动作对可能永远都不会被访问到，每次每个状态只有一个动作会被执行，通过这种方法得到的经验中，无法计算平均回报。为了克服这个问题我们引进了一个叫做exploring starts的概念。the episodes start in a state–action pair, and that every pair has a nonzero probability of being selected as the start.这个设定保证了每个状态动作对在无限组采样后都能被无限次地访问到。（显然是一个很强的设定，并非普通情况）文中还给出了一种替代的解决方向，寻求一类策略，在该策略下，理论上，任何变量的取值都师非零的。

1.5.3 Monte Carlo Control

注：这个是在exploring starts的假设下讨论的。另外，这里所说的control，是对最优策略的估计。等同于GPI中的策略提升。

因为没有模型，所以在改善策略的时候不能用到DP中的p(s',r|s,a),只能使用在MC估计当中得到的q(s,a). 假设已经产生了一系列的episodes,在策略改善时

$\pi(s) = argmax_a q(s,a)$

这一节中先给出了在exploring starts 前提下的MC control算法，核心是：存储每个状态动作对的G，在采样过程中，每经历一次S_t, A_t就在表中存储一次，并用平均值作为最新的$Q(S_t, A_t)$, 然后根据当前的Q值更新一次策略π.每个episode是一次循环。

1.5.4 Monte Carlo Control without Exploring Starts

这里在讨论如何去掉exploring starts 这个假设。有两类方法来做这件事情。一个on policy,一个是off policy. 二者的区别在于用来评估和提升的策略是否是用来产生数据的policy。

对于on-policy 的方法来讲，所使用的策略一般是 “soft",即$\pi(a|s) > 0 for all s \in S, a \in A$. 之前提到的$\epsilon - greedy$就是$\epsilon - soft$ 策略的一种。

算法框图： On-policy first-visit MC control (for ε-soft policies), estimates π ≈ π 展示了整个流程，与之前with exploring starts 的区别除了起始点随意之外，每次动作的选择也不再是应用q值最大的策略，而是改作$\epsilon - soft$ 策略。虽然也会求取一个A*但是不一定会用。另外，这里用的是A*的概率是$\epsilon/|A(s)|$，与$\epsilon - greedy$中的$1 - \epsilon$是有区别的。

在ES中，我们可以通过简单的取最大值的方法进行策略提升，是因为ES假设本身保证了所有值都能被取到，而在这里去掉了ES之后，我们不能再简单地针对当前的值函数进行贪婪探索得到策略提升，因为他会阻止对非贪婪动作的进一步探索。??

（有必要再仔细比较一下各种贪婪策略之间的关系）

1.5.5 Off-policy Prediction via Importance Sampling

问题： How can they learn about the optimal policy while behaving according to an exploratory policy?

在之前的on-policy 方法中，虽然我们明知用来估计动作值的策略不是最优策略，但是这个近似的最优的策略在探索的过程（产生数据的过程）中仍然被使用了。更直观的想法是，有两个策略，一个是被学习得到的，它最终将收敛于最优策略，我们管它叫target policy；另一个是贪婪策略，用来充分地探索整个空间，被叫做behavior policy。

那么，面临的问题就是我们要用b产生的数据去估计π所对应的状态值函数或者状态动作值函数。所以这里对b的选取有要求，即所有π能取到的值，b必须都能取到。文中叫 "$\pi(a|s) > 0 implies b(a|s)$". 当我们的策略满足这个条件时就可以采用一种用样本估计总体分布的期望值时经常用到的方法叫做重要采样。

这里重要采样率的计算方法：

$\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$

总共有两个版本，一个叫ordinary importance sampling，一个叫weight importance sampling，前者有偏，后者无偏。

$V(s) = \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T-1} G_t}{|\mathcal{T}(s)|}$

$V(s) = \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T-1} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T-1}}$

1.5.6 Incremental Implementation

这种增量技术在第二章中提到过，当时是用来计算平均的reward，而这里要用这种方法来计算平均的returns。对于ordinary importance sampling，我们把的return 已经是经过ρ缩放过的，但是针对weight importance sampling 我们还要对权重进行平均。具体的做法是，针对已经有的一系列从同一个状态出发得到的returns， $G < -\gamma G + R_{t+1}$,引入C，$C(S_t, A_t) < -C(S_t, A_t) + W$, 在更新Q 的时候用C对权重W做平均，$Q(A_t, S_t) < -Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$ W也采用增量式更新，$W < -W\frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

1.5.7 Off-policy Monte Carlo Control

将上一节的增量式更新的方法结合到off-policy的策略提升中，主要给出了MC算法的流程，基于基本的GPI框架，融合了weight importance sampling。

1.5.8 *Discounting-aware Importance Sampling

1.5.9 *Per-decision Importance Sampling

1.5.10 summary

MC 以sample episodes的形式从经验中学习值函数并优化策略，有以下三个优势：1.可以直接从与环境的交互中学习最优策略。2.可以应用到仿真或者采样的模型中。3.可以将MC方法集中在状态空间的一个感兴趣的子集当中。后续还会讨论的一个优势叫 do not bootstrap.

本章还讨论了exploring starts 的特殊性，以及on-policy 和 off-policy 的应用。

1.6 Temporal-Difference Learning

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. 也是无模型的方法，但是应用了DP中bootstrap的思想，从在估计的基础上进行估计，而不用像MC那样等到一个episodes结束。

本章要在学习TD的基础上，理清DP,MC,TD三者之间的关系。

1.6.1 TD Prediction

依然遵从GPI框架，先讨论，如果π固定的情况下，估计V_π.

TD 和MC 都是根据一组观测中非终止状态S_t的观测值，来估计$V_\pi(s)$.

MC的核心： $V(S_t) < -V(S_t) + \alpha[G_t - V(S_t)]$ 在更新时，MC只能等到episode结束才能知道G_t是什么，才能更新每个状态上的$V(S_t)$

TD的核心：$V(S_t) < -V(S_t)+\alpha[R_{t+1}+\gamma V(S_{t+1})-V(S_t)]$ 目标变为$R_{t+1}+\gamma V(S_{t+1})$ 只要等到下一时刻t+1拿到reward，就可以对当前的状态值进行更新。在这个公式中有两点是估计的而不是最优的：1.我们用当前的估计$V(S_{t+1})$取代了最优的v_π。2.我们期望的值函数是通过采样得到的，而没有遍历全空间。

这里也引入了TD-error的概念 $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$,t时刻的TD-error是在t+1时刻得到的。另外如果V在整个过程中不变，就像MC中那样，MC-error和TD-error就是等价的。

Driving home 的例子，说明了经验对下次判断的影响，以及实时的更新判断的重要性，进一步指出了MC的弱点——必须等到回合结束才能拿到return，这一点在实际应用中不一定符合场景。

1.6.2 Advantages of TD Prediction Methods

这一节梳理了TD的优势： TD 方法是基于一部分估计去估计另一部分，1.相比较于DP，TD不再需要环境的模型； 2.相比于MC，TD是在线增量式学习，MC必须等到episode结束，而TD只要等下一时刻即可； 3.虽然目前还不能用理论解释谁快，但是从应用来讲TD更快一些。

1.6.3 Optimality of TD(0)

讨论TD(0)是否具有最优性。这里还引入了所谓的batch updating。在固定α的情况下，TD 和MC都会收敛，但是收敛的值有所不同。从文中给出的收敛方差的曲线可以看出，TD收敛的值方差更小，原因是，MC仅仅在有限的轨迹上做优化，而TD则在与预测回报相关的轨迹上做优化。

1.6.4 Sarsa: On-policy TD Control

这是一个on policy 的策略提升，下一章将介绍off policy的策略。

本章中我们讨论从state-action pair 到state-action pair的转化，

$Q(S_t, A_t) < -Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$,

这里用五元组来存储每次转换$(S_t, A_t$fi$R_{t+1}, S_{t+1}, A_{t+1})$

这里之所以叫做on policy 我们从它的算法图中可以看出，每一次Q估计所用的S',A'都直接用来执行，并产生下一组数据了。

1.6.5 Q-learning: Off-policy TD Control

Q-learning 是一种off policy 的TD算法

$Q(S_t, A_t) < -Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$

对比于上面提到的Sarsa, 这里的Q是直接去估计最优的q*,与产生数据的policy独立，可以相同也可以不同。

估计Q时用的是下一个状态的最值而不是下一次要采取的动作。相比较而言，Sarsa能得到更稳健的策略，但需要更长的探索时间。

Q Learning 是off policy 方法，却没有采用重要性采样的策略。

1.6.6 Expected Sarsa

在对q*进行估计时，用期望取代了最值，这样可以消除因为随机选择动作而带来的方差。

$Q(S_t, A_t) < -Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)]$

1.6.7 Maximization Bias and Double Learning

针对的问题时，我们在前文的方法中，估计的是最优值，而实际却有可能因为greedy或者异策略而没有采用最优动作，进而造成了positive bias.

解决方案，在各种算法中引入Double 机制，行成了Double Q Learning, Double Sarsa, Double Expected Sarsa. 之所以叫double ,是因为它们在策略评估的过程中采取了两套相互独立的估计值进行估计，当然在策略提升时也是两套系统并行的。

以Double Qlearning 为例：$Q_1(S_t, A_t) < -Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, argmax_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)]$

1.6.8 Games, Afterstates, and Other Special Cases

1.6.9 summary

本章介绍了一个MC 和 DP 的结合体，TD。TD methods are alternatives to Monte Carlo methods for solving the prediction problem. In both cases, the extension to the control problem is via the idea of generalized policy iteration (GPI) that we abstracted from dynamic programming.

除了介绍基本得到TD方法之外，本章还介绍了两个常见的算法，Sarsa 和 QLearning，他们各自的优缺点及double版本。

1.7 n-step Bootstrapping

在基本理论上统一MC与TD(0),建立二者之间的纽带 n-step TD. 前文也提到过，MC是一种no bootstrap 的方法，而 TD(0)则是bootstrap的极致，所以要建立中间的过渡方法。另一种说法是，MC要等到episode结束才能进行更新，而TD（0）只要过一步就能更新，这里的N-Step TD希望能让步数更灵活地选取。另外，它也是资格迹的引子。

1.7.1 n-step TD Prediction

策略评估，这里要为n-step TD 设立一个目标。

在MC 中： $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-t-1} R_T$

在TD(0)中：$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1})$ 这里V_t是在t时刻对v_π的估计值

而在N-Step TD 中目标是：$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$

更新公式为： $V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)]$

1.7.2 n-step Sarsa

把这种n-step 思想用到策略提升当中来。有了n-step Sarsa

$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$

$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$

Expected Sarsa 只是在此基础上引入动作的加权和： $G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|S_{t+n}) Q_{t+n-1}(S_{t+n}, a)$

1.7.3 n-step Off-policy Learning by Importance Sampling

这里沿用了重要采样策略来解决off policy 问题，首先是基本off policy 的TD向n-step的扩展，$G_{t:t+n}$不变，在更新公式中加入权重，因为我们是在用给定的策略b,去近似要求优的策略π：

$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$

$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)]$ 其中，$\rho_{t:h} = \prod_{k=t} min(h, T-1) \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$,直观理解就是某个动作的从$\pi$中取值的可能性比从b中取值的可能性大，那么就要加大它所对应的回报的权重。

也有相应的 n-step Sarsa with importance sampling

1.7.4 *Per-decision Off-policy Methods with Control Variates

1.7.5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm

其实在第六章中的one-step Q Learning 和 Expect Sarsa(0) 其实也是没有用到importance sampling 的off policy, 只不过这里针对n-step TD 设计了一种替换importance sampling的方法，tree-back algorithm。

每次仍然只选择一个动作但是对于没有选择的动作对应的值仍然放入目标式子中，也就是我们的tree back 算法，不仅要包含n步执行动作的所有回报，还要包含在节点中没有执行动作的可能回报。

一步的时候：和Expected Sarsa相同

$G_{t:t+1} = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a), t < T-1$

n步时：

$G_{t:t+n} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}, t < T-1$

1.7.6 *A Unifying Algorithm: n-step $Q(\sigma)$

1.7.7 summary

本章介绍了介于TD(0)和MC之间的算法——N-Step TD, 同样从on policy 和off policy 两个方面进行展开，包括n-step sarsa ,n-step expected sarsa.

在处理off policy 的时候，提供了了两种方案，一种时重要采样，一种是tree back 算法。

1.8 Planning and learning with Tabular Methods

本章主要是对前面学习的算法进行统一和梳理。大体上分为两类，model based，如动态规划，启发式搜索；model free，如MC,TD. model based 算法以planning 为基础，model free 方法以learning 为主，本章重点加深对两个概念的理解，并尝试对两个概念进行统一整合。

In particular, the heart of both kinds of methods is the computation of value functions. Moreover, all the methods are based on looking ahead to future events, computing a backed-up value, and then using it as an update target for an approximate value function.

1.8.1 Models and Planning

model 的作用是让agent知道环境对他的动作会有什么反应。已知当前的状态和动作时，model 要对下一个时刻的状态和回报有一个预判功能。这里说的model 又可以分为分布model 和采样model两种。显然分布model更强，但是采样model往往更容易实现。 model 可以用来产生或者仿真经验。已知了初始状态和动作，model可以产生可能的transition, 当然也可能是transition 的分布。已知初始状态和策略， sample model可以产生一个完整的episode，而分布model产生所有可能的episodes。

planning 是通过模型得到策略的方法，本来也有两种。一种是state-space planning，在状态空间中搜索一条通往目标的最优路径。另一种是plan-space planning ，是以plan 为空间进行搜索，本书不涉及。那么针对于第一种State-space 的planning，这里总结了一个基本框架，这个框架在learning method中仍然适用。

model->simulated experience ->values->policy

其中有两个基本思想：(1) all state-space planning methods involve computing value functions as a key intermediate step toward improving the policy, and (2) they compute value functions by updates or backup operations applied to simulated experience.

接下来就介绍之前的各种算法皆符合这种思路，所不同的就是他们使用了不同的更新方式，更新顺序，以及保持多长时间的信息备份。

再一次指出，planning 和learning的共同点：The heart of both learning and planning methods is the estimation of value functions by backing-up update operations.

这意味着二者之间有很多可以互相借鉴的地方。不同点：The difference is that whereas planning uses simulated experience generated by a model, learning methods use real experience generated by the environment.

本章的另外一个主题是planning in small ， incremental steps 的优势。

1.8.2 Dyna: Integrated Planning, Acting, and Learning

首先给出了一个简易版的Dyna-Q,后续会对其中的各个部分进行升级以适应更普遍的环境。

model learning ， direct reinforcement learning 两种方法中，后者是通过经验直接得到value/policy，前者则是从经验中改进模型，用模型产生（planning）value/policy，也叫做indirect reinforcement learning。

二者有很多区别，但也可以进行融合。接下来给出的Dyna-Q 就借鉴了model based 的DP中的planning 和model free 的MC当中的learning。 Dyna-Q includes all of the processes shown in the diagram above—planning, acting, model-learning, and direct RL—all occurring continually.

这里用到的planning 方法是随机采样一步表格Q-Planning方法，用到的直接RL方法是一步表格Q-learning。模型学习方法也是基于表格的，并且假设环境是确定不变的。经过每一次$S_t, A_t -> R_{t+1}, S_{t+1}$之后，模型将记录这一组值。如果模型中查询到已经经历过当前状态和动作，那么它将从经验记录中返回下一状态和回报作为预测。在planning 的过程中，Q-planning 算法只从已有的经历中采样，所以模型永远不会出现查无此人的状况。

图8.2中，中间部分是一个真实的经验轨迹，左侧是一个直接的RL方法，从轨迹中提升值函数和策略。右侧是基于模型的过程。模型是从真实经历中学习得到，并且产生仿真数据。这个过程叫做search control。最后，planning是把RL方法用在仿真得到的数据上。在Dyna-Q中，同样的强化学习方法用在真实数据和仿真数据上。这个强化学习方法就成为了learning 和 planning 共有的路径。

Conceptually, planning, acting, model-learning, and direct RL occur simultaneously and in parallel in Dyna agents. 这里最耗时的是planning 的过程。具体过程见表格。

1.8.3 When the Model Is Wrong

在上一节的表格中也强调了，这里假设环境是确定的，但是在实际中并非

经常如此。发生这种情况的原因有很多，Env是随机的，采样点也只是整个空间中的一小部分，等等。这些都导致了planning 得到的是局部的最优值。如书中例三所示，如果环境突然向着利好的方向发展了怎么办，我们的agent对环境的建模还停留在原先的糟糕环境中探索。这是另一种形式的exploration与exploitation之间的冲突。这里给出了一种叫做启发式搜索的方法，其中的一种命名为Dyna-Q+。具体做法就是给那些在τ步内没有探索过的转换一个额外的reward，$\kappa\sqrt{\tau}$,当然这种探索会有很多额外的资源消耗，但是in many cases, as in the shortcut maze, this kind of computational curiosity is well worth the extra exploration. （好奇算法？）

1.8.4 Prioritized Sweeping

根据某种特殊的要求对simulation中的状态动作对儿进行排序，有计划地选择。至于按什么来排序，文中给出了一个参考——state whose value has changed。我们按照value改变量的大小对仿真的状态动作对儿进行排序。

在关于确定环境的Prioritized Sweeping算法框图中可以看到，我们一边更新Q值，一边把变化量存储起来，进行比较排序，下一轮选区初值的时候是从队列的最前端进行选取。

同样的算法可以推广到环境可变的情况中去，这里不再使用采样值进行更新，而是使用期望值进行更新。但这种替换也是有利有弊的，下一节中会进行探讨。

（但是这里并没有看到二者的真正区别在哪儿）

1.8.5 Expected vs. Sample Updates

这两种更新方法，所要更新的目标是值函数，这里系统地总结了前文方法中对各种值函数中分别用了那些方法。图中从三个维度进行总结，一个是状态值vs状态动作值，一个是以估计策略为目标更新vs以最优策略为目标进行更新，最后一个就是用期望值更新vs用采样值更新。

总结起来：期望值更新是没有采样误差的，采样值更新的采样误差是不可避免的，但是采样值更新的速度快。所以当s’空间巨大时肯定是用采样值更新，当s’计算简单时肯定用期望更新，介于中间则具体问题具体分析。

1.8.6 Trajectory Sampling

本节提供两种更新分布的方法，这里的分布应该指的时模型的分布。一种是Classical: DP中扫描每次遍历所有的状态空间，每次扫描结束会对状态空间中

的状态取值进行一次更新。很显然的问题就是，很多任务中，大量的状态存在，而且状态之间的相关性并不大，彻底遍历是没有必要的。另一种是根据某种分布去对状态空间采样，这种分布可以是均匀分布，但是并不理想，这里推荐采用on-policy分布去采样。这种分布很容易得到，而且是随时间变化的。比如，在episodic任务中，从一个起始状态一直采样到终止状态，在continue 任务中，从任意状态起始，然后一直仿真下去。这就是Trajectory的来源： one simulates explicit individual trajectories and performs updates at the state or state–action pairs encountered along the way. We call this way of generating experience and updates trajectory sampling.

那么它有什么优势呢？ both efficient and elegant! 而且从书中给出的效果图来看，状态空间越大，onpolicy的效果就越明显。

1.8.7 Real-time Dynamic Programming

首先，它的归类，是一种 on-policy trajectory-sampling version of the value-iteration algorithm of dynamic programming (DP). RTDP is an example of an asynchronous DP algorithm 。而第四章也提到了，Asynchronous 的更新是无序的，在RTDP中， the update order is dictated by the order states are visited in real or simulated trajectories.

在Sarsa中，为了得到局部优化的策略，通常需要无穷次地遍历所有的S-A对，这可以通过exploring start来保证。在这里也一样，如果是exploring starts的episodic task，RTDP是一个异步值迭代的方法，而且可以收敛到最策略。

文中指出stochastic optimal path problems更适合使用RTDP方法解决。

据说RTDP能在V->v*的时候使得π->$\pi*$

没太看懂，难道只是在原有的ADP问题中为随机选取的状态动作对加了一条轨迹？RTDP具体怎么实现的呢？

1.8.8 Planning at Decision Time

Planning 的传统应用：use planning to gradually improve a policy or value function on the basis of simulated experience obtained from a model，Selecting actions 通过其它方法来实现。Used this way, planning is not focussed on the current state. We call planning used in this way background planning.

现在想要用的地方：The other way to use planning is to begin and complete it after encountering each new state St, as a computation whose output is the selection of a

single action At; on the next step planning begins anew with St+1 to produce At+1, and so on. 用planning来做选择和决策。 Unlike the first use of planning, here planning focuses on a particular state. We call this decision-time planning.

总结：using simulated experience to gradually improve a policy or value function, or using simulated experience to select an action for the current state

1.8.9 Heuristic Search

传统的启发式搜索，每次遇到一个状态，就会考虑到一个庞大的由可能状态构成的状态树，用V估计每个分支，然后回溯到当前状态，找到最大值之后丢掉那些回溯过程中的backup的V，现在要想办法把它们存起来。通过改变值函数估计的时机和方法，来存储这些回溯之前的值。(也就是说一边估计Vt,一边改变Vt+1?)

greedy 是一个小型的（一步的）启发式搜索。现在要加深他，如果能到最后一步那是最好的，当然也会越耗费资源。这个东西在chess中用途很广泛。

1.8.10 Rollout Algorithms

定位：Rollout algorithms are decision-time planning algorithms based on Monte Carlo control applied to simulated trajectories that all begin at the current environment state.

从当前状态开始用MC方法仿真多条轨迹。具体的，从当前状态出发，根据各个可能动作以及当前的策略，得到几条轨迹，然后平均各个动作在各轨迹上得到的回报，执行估值最高的动作。

通过执行动作来估值，与MC控制的区别，不去估计全程最优的q_*或q_π,而是直接产生一个当前状态的动作值的估计，用它判断一个最优的动作，然后把这个中的东西扔掉，只保留那个最优的动作。这样更容易实现，而且不用全空间估计。

1.8.11 Monte Carlo Tree Search

定位： example of decision-time planning；is a rollout algorithm ； online ；incremental；sample-based ；从定位看是个集大成者。

以下是简要翻译，之后会做简单总结：

MCTS中，在遇到每个新的状态，并为该状态选择动作之后，这个动作是生效的；在下一个状态上通过同样的方法选择的动作也是生效的，一直这么执行

下去。在rollout 算法中，每一次执行都是一个迭代的过程，在迭代过程中要从当前的状态起一直仿真到结束状态。MCTS的核心是，连续地关注多个仿真过程，这些仿真过程都是从当前状态出发的，这些仿真是通过拓展那些在之前仿真中获得较高估值的轨迹的初始部分来实现的。尽管在很多实现中MCTS如果保存一些已经选择过的动作值会很有效果，但是这个操作并不是必须的。

对于大多数部分，在仿真轨迹中的那些动作是通过简单策略产生的，经常叫做rollout policy，因为它是针对比较简单的rollout algorithms。当rollout policy和模型都不需要太多的计算了，这些仿真轨迹就能在很短的时间内生成。在表格化的MC当中，状态动作对的值是用从该处出发的回报的平均值计算的。MC 值估计只针对状态动作对的一小部分子集，这些都是在几步之内最有可能到达的，这些状态对形成了以当前状态为根节点的树。MCTS增量式地对这棵树进行增长，增长的方式是根据仿真轨迹的结果判断哪些是更有前途的点，然后把这些点加到树上。任何仿真都会穿过这个树，然后从叶节点离开。在这棵树之外，我们用rollout policy 来进行动作选择，但是在树内可能会有更好的方法。对于这些状态，我们至少有一部分对应的动作已经有了估值，可以直接拿过来做一个明智的决策，我们管它叫tree policy,这个tree policy 可以来平衡探索和利用之间的关系。比如说，UCB,或者贪婪算法。

具体有以下四个步骤： 1.selection.根据给定的tree policy 以及每个节点上的action value，选择一条路径一直走到叶子节点。 2.expansion.在某些指定的循环中，在上一步选择的叶子节点上，通过没有探索过的动作，增加一个或者多个节点。 3.simulation.在1中选择的节点，或者2中新增加的节点上，通过rollout policy仿真，知道episode结束。 4.Backup.通过仿真轨迹得到的return会对本次经过的所有树内以及叶子节点上的动作值进行更新。终止的条件是：时间用完了，或者其它计算资源耗尽了。然后根节点的状态根据某些给定的规则以及目前该树的情况进行选择动作。也就是说每次跑一棵树只为了选一个动作而已。

最后一段列出了各种MCTS能成功的理由。

总结，MCTS每次处理一个当前状态，为了给它选择动作，从该状体开始生长一个棵树，节点是新的状态动作对，并在各个节点存储相应的return，在适当的时机对叶子节点进行生长，每次仿真在树的内部都使用tree policy，在树的叶子和外部用rollout policy进行仿真，而仿真的结果只对树内和叶子节点的状态动作值有更新，经过一定次数的循环或者到达某个条件后停止，根据当前节点上树的情况和一些具体给定的规则来选取该状态上应该采取的动作。

1.8.12 Summary

(看完第二部分之后回来做对比总结)

1.9 (Approximate Solution Methods)On-policy Prediction with Approximation

第二部分引：希望将tabular methods 拓展到更大的状态空间中去应用，由于准确最优解很难找到，所以要approximate。希望我们得到的有限经验能具有推广能力。

区别于监督学习，我们的RL虽然在function approximation中用到了监督学习的算法，但是，它具有 nonstationary,bootstrapping ,delayed targets 等特性。

行文顺序，也是从on -policy off policy 两个方面展开叙述。

第九章序

从某个已知的π产生的数据中估计v_π, 这里的新意在于，v_π不再是一个表格，而是一个以w为参数的函数。v(s,w)的形式可以是线性函数，可以是神经网络，可以是决策树，对应的w也有不同的涵义。为了达到所谓generalization的目的，通常要求w的维度要小于状态s的个数，这样有一点w的改变可以影响到很多的s,某个s的改变也会影响到其它s的值。

另外，有表格向函数的拓展，也让“部分观测问题”的解决成为了可能。

核心思想：在某个特定状态上有一个与之相对应的更新目标，估计值函数的参数不断更新，向着这个目标不断逼近。

1.9.1 Value-function Approximation

在tabular的值函数估计中，我们都会有一个目标值，比如：

MC 中，$S_t -> G_t$

TD(0) 中，$S_t -> R_{t+1} + \gamma v(S_{t+1}, w_t)$

DP 中，$S -> E_\pi[R_(t+1) + \gamma v(S_{t+1}, w_t)|S_t = s]$

在这里，我们同样把每次更新描述成一个监督问题，即，value function的输入和输出，输入时状态，或者状态动作对，输出时我们给出的一个目标值。特点是，之前的方法中，每次只更新s-v表格中的一个s对应的v,其它的是不变的，现在要更新的是s->v的函数的参数，那么每次更新相当于所有s处的v值都会改变。

另外一个特点，我们虽然可以把每次更新看作是一次监督学习，但是我们

的样本是在线得到的，所以我们用的函数估计方法，必须能够处理nonstationary targets functions.(目标函数随时间变化问题)

哪些问题有该特性，又有哪些方法可以处理呢？

1.9.2 The Prediction Objective $\bar{VE}$

针对的问题是，在某一次更新中，我们无法照顾到所有的状态，所以要为他们分配在某次更新中的权重。

总的误差：$\bar{VE}(w) = \sum_{s\in S} \mu(s)[V_\pi(s) - \hat{V}(s,w)]$

那么如何定义$\mu(s)$呢？也就是说什么样的s更重要呢？文中给出一个参考：Often $\mu(s)$ is chosen to be the fraction of time spent in s.

The on-policy distribution in episodic tasks

$h(s)$表示了不同的初始状态的可能性，$\eta(s)$表示在一个episode中s出现的次数的分布。 $\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_a \pi(a|\bar{(s)})p(s|\bar{s},a)$,其中，$h(s)$是从s起始的概率，后一项是从各个$\bar{s}$转移到s的概率。

$\mu(s) = \frac{\eta(s)}{\sum_s \eta(s)}$,归一化

另外，注：But it is not completely clear that the VE is the right performance objective for reinforcement learning. Remember that our ultimate purpose—the reason we are learning a value function—is to find a better policy. The best value function for this purpose is not necessarily the best for minimizing VE. Nevertheless, it is not yet clear what a more useful alternative goal for value prediction might be.

对VE的优化是用局部最优代替了全局最优。在接下来两节中，我们将融合强化学习方法和函数估计方法，用强化学习方法得到的更新产生的数据作为函数估计的样本。

1.9.3 Stochastic-gradient and Semi-gradient Methods

目标是极小化均方误差，这里假设v(s,w)是可微的，且每次得到的样本都服从分布μs.

SGD:$w_{t+1} = w_t + \alpha[v_\pi(S_t) - \hat{v}(S_t,w_t)]\nabla\hat{v}(S_t,w_t)$ 这里的随机体现在样本的选择上

这里的现实情况是，我们的目标并不是一个准确的值，一般都是一个估计,$v_\pi(S_t)$不能再作为目标了，这里统一用U_t来表示，U_t可以是上一次的估计值$\hat{v}$.

那么为啥会有半梯度下降呢？

如果用来得到$v_\pi(S_t)$或者U_t的方法是bootstrapping estimate,它并非真正的SGD,因为两个样本之间会有很强的关联性，文中所说w will be biased.但是这种情况下依然可以计算数据对w的影响，所以叫半SGD,它的收敛稳定性会很差，但是好处是不用等到一个episode结束。

1.9.4 Linear Methods

我们假设v是w的线性函数，这里要注意的是式子中的x(s)是特征，是feature of s.然后的梯度求解就可以了。对于SGD,这个问题的收敛性是一定的。但是针对半SGD,这种收敛性并不显然，书中有详细证明。

对比n-step semi-gradient TD 和第七章的n-step TD 的算法框图可以明显看到，基本思路是一样的，只是一个在更新w，一个是直接更新 v.

1.9.5 Feature Construction for Linear Methods

不同的特征提取方式有不同的要求，比如，对于线性方法来说，我们提取特征时特征之间要求是相互无关的，多项式特征时则要根据实际情况调节相应的阶数，此外还有基于傅里叶变换的特征，Coarse Coding，Tile Coding，Radial Basis Functions，并没有一一细看，有空要深入比较在不同的环境下如何选择合适的特征。

1.9.6 Selecting Step-Size Parameters Manually

1.9.7 Nonlinear Function Approximation: Arti□cial Neural Networks

介绍了一些ANN的基础知识，将在16章中具体讨论其应用。

1.9.8 Least-Squares TD

精确：

$w_{TD} = A^{-1}b$,其中， $A = E[x_t(x_t - \gamma x_t)^T], b = E[R_{t+1}x_t]$

估计：

$w_t = \hat{A}_t^{-1}\hat{b}_t$,其中，$\hat{A}_t = \sum_{k=0}^{t-1} x_k(x_k - \gamma x_{k+1})^T + \epsilon I$, $\hat{(b)}_t = \sum_{k=0}^{t-1} R_{t+1}x_k$

但是这里考虑到求逆的复杂度太高，所以改用增量式迭代的方法

$\hat{A}_t^{-1} = \hat{A}_{t-1}^{-1} - \frac{\hat{A}_{t-1}^{-1}x_t(x_t-\gamma x_{t+1})^T\hat{A}_{t-1}^{-1}}{1+(x_t-\gamma x_{t+1})^T\hat{A}_{t-1}^{-1}x_t}$

尽管如此，复杂度仍然是$O(d^2)$,比半梯度下降中的$O(d)$要差，它的优势在于This algorithm is the most data efficient form of linear TD(0)，而且它不用调步长α.

1.9.9 Memory-based Function Approximation

基本思路是把training examples存储下来而不更新parameters,用这些例子直接更新新的状态的值，（有点儿像监督学习里面的非参数方法，最近邻之类的），几个常见的例子：nearset neighbor;weighted average;locally weighted regression.

有点就是不用提前指定函数形式，更适合轨迹采样，避免进行全局搜索，就近更新，不会影响全局效果。但是缺点也很显然，就是存储和检索会耗时耗力。

1.9.10 Kernel-based Function Approximation

与普通的kernel方法相似，把这种方法用在了建立s与s'之间的关系上了。

1.9.11 Looking Deeper at On-policy Learning: Interest and Emphasis

之前对每个state均等看待，现在要更关注时间上比较接近的状态得到的reward。这里通过引入I_t, M_t来实现，但是没太理解这个I_t是怎么定义出来的？在11.8中又一次用到了。

1.9.12 summary

为了提高RL系统的泛化能力，把每次更新看作一次训练样本，将监督学习中的方法应用到RL中来，参数估计是常用的方法。w权重向量是我们可优化的变量，VE是均方误差我们的优化目标，SGD/semi-SGD是求解方法。特征选择的方法有：polynomial，Fourier,coars coding,titl coding,radial basis. 因为SGD要不断调节习率，所以找出一种有效的，且不需要调节学习率的方法，叫LSTD，但是目测这种方法只能用在线性函数的假设下。ANN为我们提供了足够的非线性解决方案。后面两节并没有仔细阅读。

1.10 On-policy control with Approximation

仍然服从GPI框架，策略评估，策略提升，这节讲策略提升过程中用到的on-policy approximation. 从episodic问题出发，将上一章中的state value拓展到state action value。应用到GPI框架中去，以ϵ-greedy 策略作决策。然后回到continue问题中来，对于可微的值函数，将这些思想应用到平均回报中去。

1.10.1 Episodic Semi-gradient Control

对半梯度下降进行推广，之前是$S_t -> U_t$,推广到$S_t, A_t -> U_t$ 这里的U_t可以

是MC当的G_t,也可以是n-step Sarsa 当中的returns。

半梯度下降： $w_{t+1} = w_t + \alpha[U_t - \hat{q}(S_t, A_t, w_t)]\nabla\hat{q}(S_t, A_t, w_t)$,在Sarsa(0)的情况下，$U_t = R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, w_t)$

Suitable techniques applicable to continuous actions, or to actions from large discrete sets, are a topic of ongoing research with as yet no clear resolution. 如果动作空间不是很大，我们把这个公式用到算法表中，可以看出，我们对w进行更新所用的policy就是嘴鸥采取的动作。

1.10.2 Semi-gradient n-step Sarsa

基本的n-step Sarsa 框架，先定义$G_{t:t+n}$,然后在基础的n-step Sarsa中是更新Q,现在要更新Q的参数w.

tabular 中:

$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$

$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$

Semi-graddient 版本：

$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}_{t+n-1}(S_{t+n}, A_{t+n}, w_{t+n-1})$

$w_{t+n} = w_{t+n-1} + \alpha[G_{t:t+n} - \hat{q}(S_t, A_t, w_{t+n-1})]\nabla\hat{q}(S_t, A_t, w_{t+n-1})$

1.10.3 Average Reward: A New Problem Setting for Continuing Tasks

这里既然要谈到与episodic setting 以及 discounted setting 都不同的第三种setting，那么就有必要总结一下，前两个setting都是什么有什么特点应用在什么场景下。

起源是，在continuing tasks 中，agent 与环境之间的交互既没有起点也没有终点，所以discounted setting 不适合。average reward 也是用来评价某个策略好坏的，记作$r(\pi)$

$r(\pi) = lim_{t->\infty} E[R_t|A_{0:t-1}\,\pi] = \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s',r|s,a} r$，公式当中的小r是什么？

这里要求这个马尔科夫过程具有各态历经性，这个性质保证了上面极限的存在。也就是无论从什么状态起始，无论之前经历了多少，只要策略不变，只要转移概率不变，那么经过一段时间后一定会收敛到某个状态上。$r(\pi)$是稳态下的取值，那岂不是任何状态的$r(\pi)$都相同了，而且从定义中看也应该如此，因为只与π有关，与其他的s,a都没有关系。

这里重新定义了一下回报，叫差分回报，定义为每次回报与平均回报的差值，注意这里没有回报因子一说了： $G_t = R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + ...$

利用差分回报写出来的差分值函数，以及几个贝尔曼方程,以及TD error。然后很多在tabular中用折扣回报表示的算法都可以平行推广到连续任务中，然后用平均回报来表示。在公式中做相应的替换就行了，文中举出了 the average reward version of semi-gradient Sarsa ，其它的也一样。下一次学习的时候要进行总结。

1.10.4 Deprecating the Discounted Setting

这节主要是说为啥在tabular 问题中用得好好的discounted 在我们的function approximate 问题中就不能用了。失效的主要原因是这里是一个无序的问题。

状态没有开始也没有结束，大部分用feature vector来表示，在discount中对时间顺序是有严格区分的，也就是γ的作用，然而在function approximating 问题中如果强行引进discount，当时间足够长，我们最终的计算结果会是$r(\pi)/(1-\gamma)$也同样是一个与时间顺序无关的量，所以就丢弃discount的概念。

1.10.5 Differential Semi-gradient n-step Sarsa

为了能够推广到n-step 的bootstrapping ,我们要先构造n-step 版本的TD error。（还没想清楚bootstrapping要怎么翻译）从n-step回报函数开始：

$$G_{t:t+n} = R_{t+1} - \bar{R}_{t+1} + R_{t+2} - \bar{R}_{t+2} + ... + R_{t+n} - \bar{R}_{t+n} + \hat{q}(S_{t+n}, A_{t+n}, w_{t+n-1})$$

这里的$\bar{R}_{t+1}$是对$r(\pi)$的估计，但问题是$r(\pi)$本来都一样，但是估计它的$\bar{R}_{t+1}$fi$\bar{R}_{t+2}$好像并不一样啊？在算法框图中写成了$\bar{(R)}$?

TD error 定义为 $\delta_t = G_{t:t+n} - \hat{q}(S_{t+n}, A_{t+n}, w_{t+n-1})$

然后就可以应用到相应的算法中去了，注意对比semi-gradient n-step Sarsa 。

1.10.6 Summary

本章中我们把前面一章中提到的参数估计的思想和半梯度下降思想拓展到控制问题。（这个半梯度的定义到底是针对什么东西说的呢）

我们在连续任务的函数估计问题中引入了一个叫做平均回报的东西来代替表格问题中折扣因子的作用，然后根据平均回报定义了差分值函数，差分TD误差，以及差分回报形式的贝尔曼方程。这些基础概念进行重新定义之后，在具体的算法中平行的推进就可以了。

1.11 off-policy Methods with Approximation

treat off-policy case with function approximation.6,7章中的off policy 方法可以很容易的扩展到半梯度下降方法中，但是，这些方法的收敛性都不如onpolicy 那样好。本章将先讨论一些易于学习的方法，然后再有着强收敛性保证的方法。在结束的时候还会讨论一些改进方法，但是无论从理论上还是实践上，和on policy 的拓展相比，这些方法都不是很完美。

在之前学过的off policy问题中，我们根据behavior 策略b产生的数据，估计目标策略π的值函数。那个时候，两个策略都是已知且不变的。在control case，两个策略都是随着学习的进程而改变的，比如，π是随着$\hat{q}$而改变的贪婪策略,b也是随着$\hat{q}$而改变的ϵ。拓展到函数估计上会有两个困难的问题：1.更新的目标（目标函数的构造）。2.更新的分布。前文书提到的重要性采样，只能解决第一个问题，会有采样偏差，但是解决了大部分问题，在本章中第一部分也会介绍。而对于第二个问题，在off policy 问题中的更新的分布与on policy 问题中的并不一致，而这种相同的策略对半梯度下降方法的稳定性是很重要的。这里又提出两个方法来解决这个问题：第一种是再次使用重要性采样，让半梯度方法收敛。第二个是构造一种真正意义上的梯度下降，不再依赖于某个特殊的分布。 This is a cutting-edge research area, and it is not clear which of these approaches is most effective in practice.

重要采样用来解决什么问题的？

1.11.1 Semi-gradient Methods

We begin by describing how the methods developed in earlier chapters for the o -policy case extend readily to function approximation as semi-gradient methods.

这是针对off-policy 的第一个挑战——changing the update targets.这些算法经常不收敛。虽然在表格学习中是完全可以保证稳定性的。

为了把第7章中的off policy算法转换成半梯度下降问题。这里同样需要把V和Q 换成w, 也需要估计函数 $\hat{v},\hat{q}$ 和它们的梯度，对重要采样的比率仍然定义为：

$$\rho_t = \rho_{t:t} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

之后就和之前对应的算法一样了，这里不进行一一列数。有空把前面的整理成表格，然后统一列到这里来。

1.11.2 Examples of Off-policy Divergence

举了两个例子，说明了off-policy 用来做函数估计时带来的权重发散问题。

1.In the on-policy case the promise of future reward must be kept and the system is kept in check. But in the off-policy case, a promise can be made and then, after taking an action that the target policy never would, forgotten and forgiven.

2.This example is striking because the TD and DP methods used are arguably the simplest and best-understood bootstrapping methods, and the linear, semi-descent method used is arguably the simplest and best-understood kind of function approximation. The example shows that even the simplest combination of bootstrapping and function approximation can be unstable if the updates are not done according to the on-policy distribution.

1.11.3 The Deadly Triad

Our discussion so far can be summarized by saying that the danger of instability and divergence arises whenever we combine all of the following three elements, making up what we call the deadly triad:以下三种情况如果同时出现的话，可能会面临不稳定的情况：

Function approximation: A powerful, scalable way of generalizing from a state space much larger than the memory and computational resources (e.g., linear function approximation or artificial neural networks). 显然函数估计是不可以舍弃的，为了处理大规模问题进行函数估计是必要的，诸如LSTD等方法复杂度太高(平方复杂度)，无法使用。

Bootstrapping: Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusively on actual rewards and complete returns (as in MC methods).这种方法有利于节约计算和存储成本，也是一种在大规模问题中不可或缺的手段。

Off-policy training: Training on a distribution of transitions other than that produced by the target policy. Sweeping through the state space and updating all states uniformly, as in dynamic programming, does not respect the target policy and is an example off-policy training. For model-free reinforcement learning, one can simply use Sarsa rather than Q-learning.但是据说在后续的应用中会有很广泛的需求。比如在智能体的多任务并行学习中，单一的behavior 策略不可能满足所有的任务需求，

所以用到off -policy 也是必然的。

三个问题中如果只又两个，我们是可以解决的，但是三个都存在要怎么办？

1.11.4 Linear Value-function Geometry

首先来看一个三个状态，两个权重的例子，我们用函数估计将数据的表示降维。假如π的最佳函数v_π很难表示出来，那么我们要在子空间上为它找个替代品，可是怎么知道替代品和最佳的函数相似度呢？这里定义一个带权重的欧式距离来形容二者之间的远近。

这里定义了一个贝尔曼误差，贝尔曼误差是什么？与TD误差有什么区别？与最开始提到的投影又是什么关系？贝尔曼误差是如何描述值函数估计问题的？

贝尔曼误差最初是定义在某个状态上的：$\bar{\delta}_w(s) = (\sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r+\gamma v_w(s')]) - v_w(s)$,期望的形式$\bar{\delta}_w(s) = E[R_{t+1} + \gamma v_w S_t | S_t = s, A_t\ \pi]$

前文提到了TD-error : $\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t) - \hat{v}(S_t, w_t)$

可见贝尔曼误差是TD误差的期望。

如果是在DP的问题中，B误差将收敛到一个最优策略上，即$v_\pi = v_\pi B_\pi$; 但是在函数估计问题中，这段没看太懂，大致意思就是这种情况下无法在高维空间找到相应的停留点，整个过程只能在子空间进行，但具体为啥，没想明白。

针对子空间，也就是投影空间，求取所谓的Mean Square Projected Bellman Error $P\bar{B}E$,多了一个投影变换 $\prod$, $P\bar{B}E(w) = || \prod \delta_w ||^2_\mu$

$P\bar{B}E$收敛的点竟然和TD收敛的点相同，至于为啥，不知道。

这里先用one-step TD error 作为起点来构造真正的SGD算法，构造的结果是：$w_{t+1} = w_t + \alpha\rho_t\delta_t(\nabla\hat{v}(S_t, w_t) - \gamma\nabla\hat{v}(S_{t+1}, w_t))$

对比于半梯度 $w_{t+1} = w_t + \alpha\rho_t\delta_t\nabla\hat{v}(S_t, w_t)$,只差了最后一项，文中举出一个例子来说明这看似合理的东西在简单的情况下是如何失效的。例子说明了TDE最小并不一定就是最好的。the naive residual-gradient algorithm finds different values, and these values have lower TDE than do the true values.

之后又举了一个A-presplit的反例说明了BE也无法得到有效的值函数。 This example shows intuitively that minimizing the BE (which the residual-gradient algorithm surely does) may not be a desirable goal.

1.11.5 Gradient Descent in the Bellman Error

回到off-policy 做函数估计时的稳定性上来。

这里有句话道明了这几个部分之间的关系。一定要认真理清楚。 Although this has been a popular and influential approach, the conclusion that we reach here is that it is a misstep and yields no good learning algorithms. On the other hand, this approach fails in an interesting way that offers insight into what might constitute a good approach。说得是，这里讲述的方法是很流行的，但是我们将要证明它是错的，然后我们还要证明尽管是错的，也会有可用的地方，它对我们后面的算法是很有用的。那么我们要做的首先是去了解这个方法到底是什么？为什么就错了？又对后文有什么启发？

1.11.6 The Bellman Error is Not Learnable

所谓的Learn able是说我们能不能在多项式时间内得到结果。这里放宽了限制，放到无限次样本，但结论却证明，即使是无限次，bellman error 也是 not Learn able。这一章虽然说明了BE不好用，但却引出了一个叫PBE的东西Projected Bellman Error

1.11.7 Gradient - TD Methods

接下来就在$P\bar{B}E$的基础上进行梯度下降算法的推导（ true SGD ）。文末的评语：Gradient-TD methods are currently the most well understood and widely used stable off-policy methods。

基本推导思路：先把它的具体形式写出来，然后求导，在求导过程中用样本的期望来代替式子中的各个因子，但是这三个部分之间是有相关性的，都与下个状态的特征向量x_{t+1}有关，我们不能对两个期望直接采样然后相乘，（为啥），那样会让我们的对梯度的估计产生偏差，在naive residual-gradient algorithm中就存在这个问题。（为啥没有发现）

另一种思路是把它们三个拆开来估计，然后再结合到一起去，产生一个有偏差的梯度估计。能够工作但需要很多的计算资源。把这个想法做一个改进，三个期望中两个进行估计，一个进行采样，但是不幸的是，整体的复杂度仍然有d^2。但是这种想法被延续下来了。

在Gradient - TD中，估计和存储后两个因子的乘积，前两项的乘积是个d维的向量，和w同维度。

$v \approx E[x_t x_t^T]^{-1} E[\rho_t \delta_t x_t]$ 公式11.28中所代表的具体含义是什么？

GTD 和 TDC的区别难道不是在于是否把括号打开吗？

GTD2: $w_{t+1} = w_t + \alpha E[\rho_t(x_t - \gamma x_{t+1})x_t^T)]E[x_t x_t^T]^{-1}E[\rho_t \delta_t x_t] = w_t + \alpha\rho_t(x_t - \gamma x_{t+1})x_t^T v_t$

TDC: $w_{t+1} == w_t + \alpha E[\rho_t(x_t - \gamma x_{t+1})x_t^T)]E[x_t x_t^T]^{-1}E[\rho_t \delta_t x_t] = w_t + \alpha\rho_t(\delta x_t - \gamma x_{t+1} x_t^T v_t)$

它们都包含两个学习的过程，一个是对w的学习，另一个是对v的学习。

1.11.8 Emphatic - TD Methods

接下来讨论第二个得到廉价且高效的在函数估计中使用的off policy 学习方法。

另一种理解γ的方式，把discounting 看作一种伪终点，没理解为啥这种方式能够在即使随意选起点，而且不终止的不重启的情况下异策略也能收敛。

Emphatic -TD, 好像前面9.10讲过。但是这件事情究竟解决了异策略时的什么问题呢？

结尾：The algorithm converges to the optimal solution in theory on this problem, but in practice it does not.

1.11.9 Reducing Variance

off policy 本来就比on policy 的方差要大，文中的比喻说，通过下厨做饭来学开车是不太现实的。那么我们为啥要控制重要采样的方差呢？或者叫偏差。因为ρ的存在，彼此不相关，可高可低，乘积便会产生更大的偏差。另外这个ρ还乘在了SGD的stepsize的α上了。也就是每步的步幅相差也很大。（具体公式），在第五章中，用权重的重要采样来解决tabular 问题，方差也很小，在值函数估计中也有用到。（参考文献）。在第七章中也提到了可以用tree back的方法来避免使用重要性采样。

1.11.10 summary

why "off policy"？ 第一，要平衡探索和利用；第二，要避免目标策略的独断。

how to do？ 1.用表格学习的方法会有偏差，2.用含有bootstrap的半梯度法会出现不稳定的现象。在实际应用中会有3个不能同时有的问题解决方法：最初尝试用贝尔曼误差，来实现真正的SGD，但是事实证明在特征维度上的学习并不

可靠，要应用的状态维度才行，然后就放弃了。将投影引入，形成了GTD算法。最后又把之前的Emphatic-TD拿过来了。

1.12 Eligibility Traces

TD(λ)中的也是Eligibility trace的一种应用。而Q-Learning 和Sarsa 也是TD的一种，也可以和Eligibility trace相结合。

TD+ET产生了一类介于TD(0)和MC之间的方法。ET是连续问题中在线MC的实现。在线MC是什么？

虽然介于TD(0)和MC之间，但这并不是n-step TD, 这里提供的是一种更有力的计算机制，要将n-step TD 与TD(λ)作区分。

基本思想，当w_t的一部分加入到估计值的产生中时，z_t增加，然后慢慢消失，在其消失之前学习相应的w_t,有一个参数用来控制z_t的消失速率。

最基本的计算优势：相比于n-step TD. 1.只存储一个trace vector。而不是last n 个特征向量。 2.learning 在时间上是连续的，且3.没有delay.

另外一个核心：forward view VS backwards view。最开始在TD learning的时候提过，但要找到它们的不同。

本章展开顺序：$statevalue\&prediction -> state-actionvalue\&control -> onpolicy -> offpolicy$.

1.12.1 The λ-return

这是个基础概念.在 tabular中有n-step return ： $G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, w_{t+n-1})$，是SGD等算法的更新目标。现在的想法是，把每一步的n-step return 做个平均，或者给每个回报加个权重，再对权重归一化一下，就得到了下面的λ-return:

$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$

在episode 的时候最后一项就变成 $\lambda^{T-t-1} G_t$ 为啥？

$\lambda = 0$是1-step return ,$\lambda = 1$是平均的MC回报。

第一个算法：off-line λ-return algorithm.

公式12.4

这里离线的意识是在整个episode中权重并不是实时改变，每个episode结束后更新一次w_{t+1}.但是从其目标来看，这是一个向前看的算法，并不具有真正的eligiblility trace 的性质。

1.12.2 TD-λ

TD(λ)是off-line λ-return algorithm 的拓展，优势在于，1.每一步更新w_t,2.计算平均分配到每个时间点上，3.能拓展到连续任务中去。

实现方法：引入于w_t同维度的z_t，这个z_t肯定要比episode要短，w_t负责关注长时间跨度的记忆，z_t负责短时记忆，他直接作用于w_t，间接改变value。

定义：$z_t = \gamma\lambda z_{t-1} + \nabla\hat{(v)}(S_t, w_t)$

$\gamma\lambda z_{t-1}$折扣因子和回报系数的乘积，均在0-1之间，所以这是个衰减项目。$\nabla\hat{(v)}(S_t, w_t)$是在跟踪value的变化，形成反馈，相当于PID中的D.

$w_{t+1} = w_t + \alpha\delta_t z_t$

注意这里的δ_t是TD error，不是 G_t^λ,所以不用整个episodic 走完。z_t是在看过去，δ_t是在看未来。

当$\lambda = 0$时，就是Semi-Gradient TD in Tabular. 随着γ衰减的TD，也是MC的一种更普遍的形式。相比于MC，他不受回合限制，相比于TD,incrementally,online.防止坏点出现？

另外，这个算法收敛，但不能收敛到最小误差。

1.12.3 n-step Truncated λ-return Methods

指出 offline λ-return algorithm 是理想情况，episode没结束时， G_t^λ就不知道。但是在eligibility trace的时候rλ会把末尾一段衰减掉，这种近似一段的思想被直接应用到实践中了。

用h代替12.3中的T.得到：

用$G_{t:h}^\lambda$作为目标，叫做TD(λ).

$w_{t+n} =$

这里还提到了一个叫K-step λ-return 的算法，跟上个式子很像，是比较二者区别。

1.12.4 Redoing Updates: The Online λ-return Algorithm

之前的算法时offline 算法，n要大，才能有效近似offline λ return。n 还要小，才能更快更新，更早影响到下一步动作。那么改如何平衡？

理想方案，没出现一组新的increment，回去更新所有来时的路，这样既能近似n-step truncated λ-return, 又能实时更新视野，但这样每次都产生w1,w2,...,wt为了区分它们，重新标记，并排成三角。

根据更新公式,这已经是一个online 算法了。但是计算复杂。

w_{t+1}^h =

1.12.5 True Online TD(λ)

这里的true 是相对于TD(λ)而言的。只用到了更新过程中的w_t^t，为啥？

更新公式

w_{t+1} =

其中 δ_t =

z_t =

这种叫做dutch trace :dutch s是aa均摊的意思。

1.12.6 Dutch Traces in Monte Carlo Learning

这一节最终的结论好像并没有比MC好到哪里去，只是说明了资格迹这种东西不仅能从TD的角度推演过来，也可以从MC的角度推演。

wt 本来是在每个episode结束才更新的，现在也是这样，但是把中间结果的计算放在每个步骤中来。Wt写成一种增量形式，计算两个复杂度为d的辅助变量。

具体如下：

。

1.12.7 Sarsa(λ)

与TD不同的是，这里把v换成q了。其它的步骤都一样。

G:

λ-return:

w_{t+1} =

1.12.8 Variable λ and γ

这一节是on policy 的总结阶段，更general 的形式，每个时刻t 对应了不同的 λ_t γ_t,

Gt=

.

$\lambda - return$ 为：

。

其中λ_s的含义是从state value 得到的提升，同样可以从action state value 出发。

。

1.12.9 Off-policy Eligibility Traces with Control Variates

把上文中关于$G_t^{\lambda_s}$的定义改一下，写成如下形式：

$G_t^{\lambda_s}$ =

为啥会有两部分？

下面出现一个forward view 这里要重点理解什么是forward 什么是backwards

w_{t+1} =

。其中变量的含义：

这里还给出了一个on policy 和 off policy 共用的z_t

z_t

由重要采样系数决定是on policy 还是off policy。但是不稳定，是一个半梯度法，那么之前的问题又出现了，到底什么是半梯度？

1.12.10 Watkins’s Q(λ) to Tree-Backup(λ)

从 Q-learning 到 eligibility trace，叫 Watkins’s Q(λ)。把资格迹延迟到greedy action 的步骤，在非贪婪策略出现时停止。

据说在第六章中，曾经将Q-learning和E Sarsa 统一过，可以查查。

1.12.11 Stable Off-policy Methods with Traces

基于11章中的Gradient TD Emphatic-TD

1.12.12 Implementation Issues

1.12.13 conclusion

1.13 Policy Gradient Methods

1.13.1 Policy Approximation and its Advantages

1.13.2 The Policy Gradient Theorem

1.13.3 REINFORCE: Monte Carlo Policy Gradient

1.13.4 REINFORCE with Baseline

1.13.5 Actor–Critic Methods

1.13.6 Policy Gradient for Continuing Problems

1.13.7 Policy Parameterization for Continuous Actions

1.13.8 summary

（对比总结第一部分第二部分）

第2章　模式识别课程笔记

2.1　模式识别与机器学习概论

2.1.1　模式、模式识别、机器学习

什么是模式？

什么是模式识别？

典型过程是什么？

模式识别与机器学习？ ML 偏向数学理论 RL 偏向应用，设计模型的过程应用的是机器学习方法 DM 偏向数据应用，非感知数据

统计学习方法包括哪些？

期望风险最小化vs 经验风险最小化

分类损失vs回归损失vs 概率密度估计

总结分类器的设计过程

样本数目不足以进行测试验证时要进行交叉验证

？验证与识别与检测有何区别？

一般分类问题的指标有哪些？

两类问题中的指标有哪些？

ROC曲线？

多类问题的召回率和精度？

经验误差与期望误差

泛化界

VC维泛化界中的h

什么时过拟合

统计方法vs结构方法

生成模型vs判别模型

用判别模型的准则估计生成模型的参数

break

贝叶斯决策的表示

soft-max

对公式，要联系其物理意义，或者是头脑中的直觉

为什么贝叶斯分类器是最小错误率分类器？

参数估计vs非参数估计

贝叶斯估计和最大似然估计

梯度下降，梯度上升

递归贝叶斯学习，增量式估计

回归模型的贝叶斯估计

2.2 统计模式识别

2.3 判别学习

2.4 集成学习

2.5 特征提取与特征选择

2.6 神经网络与深度学习

2.7 核方法

2.8 图模型

2.9 度量学习和半监督学习

2.10 迁移学习和多任务学习

2.11 结构模式识别