Reinforcement Learning: An Introduction notebook

黎雷蕾

2018 年 1 月 25 日

目录

Chapter 1

Introduction

Chapter 2

Multi-armed Bandits

Chapter 3

Finite Markov Decision Processes

Chapter 4

Dynamic Programming

Chapter 5

Monte Carlo Methods

Chapter 6

Temporal-Difference Learning

6.1 TD Prediction

时序差分算法 (TD) 和蒙特卡洛算法 (MC) 都是根据策略 π 进行模拟，根据当前状态 S_t 和 v_π 来更新相应的 V 值.

α 步长-MC 算法可以表示为:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right] \tag{6.1}$$

其中 G_t 表示在时间 t 时整个模拟的回报，α 是一个恒定步长参数 (constant step-size parameter)。上述公式表示 MC 算法必须在一次完整的模拟之后 (必须得到 G_t) 才能更新 $V(S_t)$。

TD 算法只需要等待下一步，在 $t+1$ 时立即用观察到的 R_{t+1} 和估计的 $V(S_{t+1})$ 进行更新:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{6.2}$$

上式可以被称为 TD(0)，或者 one-step TD。算法可以表示为:

TD(0) for estimating v_π

1. IN: 需要估计的策略 π;
2. 初始化 $V(s) = 0$;
3. 重复多次轨迹采样直到达到结束条件:
   (a) 初始化 S;
   (b) 通过当前的 S 和所给的策略 π 来更新动作 A;
   (c) 通过动作 A, 观察得到 R_{t+1} 和 S_{t+1}，并根据公式 6.2更新 $V(S)$;
   (d) $S \leftarrow S_{t+1}$;

TD(0) 是 bootstrapping 的算法，即:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] \tag{6.3}$$

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \right] \tag{6.4}$$

其中 6.3是 MC 的目标而 6.4是 TD 的目标。

注意到当前的估计状态值 $V(S_t)$ 与更好的估计 $R_{t+1} + \gamma V(S_{t+1})$ 会存在一定的差异，我们将其称为 $TD\ error$，用 δ_t 表示:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{6.5}$$

注意到 δ_t 的产生只与当前时间 t 和下一时间 $t+1$ 有关，如果 V 在某次估计中没有进行改变，那么蒙特卡洛 error，可以表示为:

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \end{aligned} \tag{6.6}$$

6.2 Advantages of TD Prediction Methods

TD 算法是通过 bootstrap 进行迭代的。与 MC 算法比较如下:

- MC 具有高方差 (variance)，零偏差 (bias)；
  - 具有很好的收敛 (convergence) 性质 (可以采用函数逼近)；
  - 对于初始化的值不敏感；
  - 非常简单易用；
- TD 具有低方差和偏差；
  - 一般来说比 MC 效率高；
  - TD(0) 收敛于 $v_\pi(s)$(但不是总能采用函数逼近)；
  - 对初值比较敏感；

6.3 Optimality of TD(0)

批量更新 (batch updating): 我们可以理解为当处理完一批数据后再进行更新，而不是每处理一个数据就进行更新了。

假设第 k 次模拟的蒙特卡洛轨迹为：

$$\langle s_1^k, a_1^k, r_2^k, \cdots, s_{T_K}^k \rangle \tag{6.7}$$

- MC 收敛于最小均方误差, 契合观察到的结果：

$$\sum_{k=1}^{K}\sum_{t=1}^{T_K}(G_t^k - V(s_t^k))^2 \tag{6.8}$$

- TD 收敛于最大似然马尔可夫模型，得到的 MDP$\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \rangle$ 切合数据：

$$\begin{aligned} \hat{\mathcal{P}}_{s,s'}^a &= \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_K}\ell(s_t^k, a_t^k, s_{t+1}^k = s, a, s') \\ \hat{\mathcal{R}}_s^a &= \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_K}\ell(s_t^k, a_t^k = s, a) r_t^k \end{aligned} \tag{6.9}$$

通过上面的分析，TD 更适合马尔可夫环境，MC 更加适合非马尔可夫环境。

6.4 Sarsa: On-policy TD Control

TD(0) 的 Q 值可由如下动作价值公式进行更新:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (6.10)$$

如果到 S_{t+1} 时发生中断，那么 $Q(S_{t+1}, A_{t+1}) = 0$;

sarsa 算法采用五元组 $\langle S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1} \rangle$ 进行迭代, 对应下图:

图 6.1: sarsa 算法示意图

对应的算法可以表示为:

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

1. 初始化 $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$ 随机或者为 0，$Q(terminal-state, \cdot) = 0$;
2. 重复多次轨迹采样直到 S 到达中断状态:
   (a) 初始化 S;
   (b) 采用 ϵ-贪心算法从 Q 中获取 A, S;
   (c) 对于每条轨迹的每一步 (each step in one episode), 进行如下循环:
      i. 通过选取的 A，观察 R S‘;
      ii. 根据 S' 从 Q 中通过 ϵ-贪心算法选取 A';
      iii. 更新 $Q(S,A)$:

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)];$$

      iv. 更新 $S \leftarrow S'$;
      v. 更新 $A \leftarrow A'$;

6.5 Q-learning: Off-policy TD Control

$Q-learning$ 通过下面的公式来更新 $Q(S_t, A_t)$:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)] \quad (6.11)$$

对应的算法可以表示为:

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

1. 初始化 $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$ 随机或者为 0，$Q(terminal-state, \cdot) = 0$;
2. 重复多次轨迹采样直到 S 到达中断状态：
   (a) 初始化 S;
   (b) 采用 ϵ-贪心算法从 Q 中获取 A, S;
   (c) 对于每条轨迹的每一步 (each step in one episode), 进行如下循环:
      i. 通过选取的 A，观察 R S‘;
      ii. 根据 S' 从 Q 中通过 ϵ-贪心算法选取 A';
      iii. 更新 $Q(S,A)$:

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$$

      iv. 更新 $S \leftarrow S'$;

6.6 Expected Sarsa

对于类似 $Q-learning$ 之类的算法，如果不是采用最大值，而是采用期望值，即照着下面的公式进行更新：

$$\begin{aligned} Q(S_t,A_t) &\leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1},A_{t+1})|S_{t+1}] - Q(S_t,A_t)\right] \\ &\leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1},a) - Q(S_t,A_t)\right] \end{aligned} \tag{6.12}$$

这个算法被称为 Expected Sarsa, 相比于 Sarsa，Expected Sarsa 计算将会更加复杂，但是由于 A_{t+1} 采用的是随机选择而不是最大化，能够有效地降低方差。

6.7 Maximization Bias and Double Learning

前面六节所提到的所有算法在优化的过程中都需要最大化目标策略，但是这样做很容易导致一个最大化偏差 (Maximization Bias)。

想要避免最大化偏差，若我们一直使用估计的最大值作为真实值的估计 $(Q-learning)$，那么会产生一个正的偏差，造成这个的原因是由于在确定最大化 Q 值的动作和估计状态值时采用的是相同的样本。

为了解决这个问题，我们把样本分成两个子集 (set)，分别独立地学习两个估计，分别称为 $Q_1(a)$ 和 $Q_2(a)$, 我们可以用 $Q_1(a)$ 来决定最大化动作：$A^* = \arg\max_a Q_1(a)$，采用 $Q_2(a)$ 对其价值函数进行估计：$Q_2(A^*) = Q_2(\arg\max_a Q_1(a))$. 若 $\mathbb{E}[Q_2(A^*)] = q(A^*)$。那么将会是无偏估计。我们还可以产生第二个无偏估计 $Q_1(\arg\max_a Q_2(a))$,这个算法被称为 $doubled\,learning$。这个算法需要两倍的存储空间，但是每一步的计算开销没有增加。下面给出的是 $Q-learning$ 的 double 版：

Double Q-learning

1. 随机地初始化 $Q_1(s,a), Q_2(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$;
2. 初始化中断状态: $Q_1(terminal-state,\cdot) = Q_2(terminal-state,\cdot) = 0$;
3. 对于每次估计 (episode):
   (a) 初始化 S;
   (b) 对于当前估计的每一步:
      i. 在 Q_1+Q_2 中采用 ϵ-贪心算法从 S 中获取 A;
      ii. 通过选择的 A，观察 R, S':
      iii. 以 0.5 的概率随机选择下面两个公式其中的一个:

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\left(R + \gamma Q_2(S', \arg\max_a Q_1(S',a)) - Q_1(S,A)\right)$$
$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\left(R + \gamma Q_1(S', \arg\max_a Q_2(S',a)) - Q_2(S,A)\right)$$

      iv. 更新 $S \leftarrow S'$

6.8 Games, Afterstates, and Other Special Cases

在游戏模拟中，可能从不同状态选取不同的动作，最后得到的状态是相同的，这被称为 Afterstates。Afterstates 在我们知道部分初始环境的变化的情况后比较有用，我们不需要知道全部的动态，在 Afterstates 中，Q 值按照如下方式更新:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)] \quad (6.13)$$

Chapter 7

Multi-step Bootstrapping

7.1 n-step TD Prediction

蒙特卡洛方法就是一个 n 步的 TD 算法。one-step return 可以写为：

$$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1}) \tag{7.1}$$

two-step return 可以写为：

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}) \tag{7.2}$$

那么 n-step return:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \tag{7.3}$$

n-step TD for estimating $V \approx v_\pi$

1. 随机地初始化 $V(s)$;
2. 准备两个参数：步长参数 $\alpha \in (0, 1]$, 一个正整数 n;
3. 对于每次估计：
   (a) 初始化 S_0 并使其不处于中断状态;
   (b) $T \leftarrow \infty$;
   (c)
   for $t = 0, 1, 2, \cdots$ & $\tau \neq T - 1$ **do**
     if t<T **then**
       根据 $\pi(\cdot|\pi)$;
       观察并记录 R_{t+1} 和 S_{t+1};
       if S_{t+1} 为中断状态 (terminal) **then**
         $T \leftarrow t + 1$
       end if
     end if
     $\tau = t - n + 1$; 表示将要更新的阶段的下标;
     if $\tau \geq 0$ **then**
       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$;
       if $\tau + n < T$ **then**
         $G \leftarrow G + \gamma^n V(S_{\tau+n})$;
       end if
       $V(S_\tau) \leftarrow V(S_\tau) + \alpha[G - V(S_\tau)]$
     end if
   end for

7.2 n-step Sarsa

我们首先可以定义 n 步动作函数 Q 的 return：

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \quad (7.4)$$

Q 值可以被定义为：

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)] \quad (7.5)$$

那么 n-step Sarsa 可以被定义成：

n-step Sarsa for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

1. 随机地初始化 $Q(s,a)$;
2. 根据 Q 采用 ϵ-贪心算法初始化 π，或者根据实际需求初始化策略 π;
3. 参数: 步长参数 $\alpha \in (0,1]$，$\epsilon > 0$，一个正整数 n;
4. 对于每次模拟，均进行如下操作:
   (a) 初始化 S_0 使其不处于中断状态;
   (b) 选择动作 $A_0 \sim \pi(\cdot|S_0)$;
   (c) $T \leftarrow \infty$;
   (d)

```
for t = 0, 1, 2, ··· && τ ≠ 0 do
    if t < T then
        选择动作 A_t;
        根据动作 A_t 观察得到下一个 R_{t+1}, S_{t+1};
        if S_{t+1} 中断状态 then
            T ← t + 1;
        else
            选择动作 A_{t+1} ~ π(·|S_{t+1});
        end if
    end if
    当前时序下标:τ ← t − n + 1;
    if τ ≥ 0 then
        G ← Σ_{i=τ+1}^{min(τ+n,T)} γ^{i−τ−1} R_i;
        if τ + n < T then
            G ← G + γ^n Q(S_{τ+n}, A_{τ+n});
        end if
        Q(S_τ, A_τ) ← Q(S_τ, A_τ) + α[G − Q(S_τ, A_τ)];
        if 策略 π 已经被学习到 then
            确保 π(·|S_τ) 是关于 Q 的 ε-贪心法;
        end if
    end if
end for
```

我们可以将 return 定义成：

$$G_{t:t+n} = R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|S_{t+n}) Q_{t+n-1}(S_{t+n}, a) \quad (7.6)$$

这样就可以把上述算法改写成 Expected Sarsa 算法了。

7.3 n-step Off-policy Learning by Importance Sampling

在异策略 TD(n) 算法中引入重要性采样，并设定权重 $\rho_{t:t+n-1}$，那么 return 可以重新定义成：

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)] \quad (7.7)$$

其中权重 ρ 可以定义为：

$$\rho_{t:h} = \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \quad (7.8)$$

同理 7.2 章的 n-step Sarsa 的更新公式也可以写成：

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n-1}[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)] \quad (7.9)$$

那么 off-policy n-step Sarsa 算法可以写作：

Off-policy n-step Sarsa for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

1. **输入**: 一个随机的策略 b，只需保证 $b(a|s) > 0$ 即可；
2. 随机地初始化 $Q(s, a)$;
3. 根据 Q 采用 ϵ-贪心算法初始化 π，或者根据实际需求初始化策略 π;
4. 参数: 步长参数 $\alpha \in (0, 1]$，$\epsilon > 0$，一个正整数 n;
5. 对于每次模拟，均进行如下操作：

(a) 初始化 S_0 使其不处于中断状态;

(b) 选择动作 $A_0 \sim \pi(\cdot|S_0)$;

(c) $T \leftarrow \infty$;

(d)

for $t = 0, 1, 2, \cdots$ && $\tau \neq 0$ **do**
  if $t < T$ **then**
    选择动作 A_t;
    观察并记录 R_{t+1} 和 S_{t+1};
    if S_{t+1} 是中断状态 **then**
      $T \leftarrow t + 1$;
    else
      选择动作 $A_{t+1} \sim \pi(\cdot|S_{t+1})$;
    end if
  end if
  当前时序下标:$\tau \leftarrow t - n + 1$;
  if $\tau \geq 0$ **then**
    $\rho_{\tau+1:t+n-1} \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$;
    $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$;
    if $\tau + n < T$ **then**
      $G_{\tau:\tau+n} \leftarrow G_{\tau:\tau+n} + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$;
    end if
    if 策略 π 已经被学习到 **then**
      确保 $\pi(\cdot|S_\tau)$ 是关于 Q 的 ϵ-贪心法;
    end if
  end if
end for

7.4 Per-reward Off-policy Methods

前三节提到的 n-step 算法结构简单，但是可能效果并不是最好的。比较好的方法是采取 Per-reward 算法，它的 return 可以被定义为：

$$G_{t:h} = R_{t+1} + \gamma G_{t+1:h} \tag{7.10}$$

同样地，我们定义权重 ρ:

$$\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)} \tag{7.11}$$

off-policy 的 return 可以写为：

$$G_{t:h} = \rho_t(R_{t+1} + \gamma G_{t+1:h}) + (1 - \rho_t)V(S_t) \tag{7.12}$$

改成 Q 值：

$$\begin{aligned} G_{t:h} &= R_{t+1} + \gamma(\rho_{t+1}G_{t+1:h} + (1 - \rho_{t-1})\overline{Q}_{t+1}) \\ \overline{Q}_{t+1} &= \sum_a \pi(a|S_t)Q_{t-1}(S_t, a) \end{aligned} \tag{7.13}$$

采用 off-policy 要注意：

1. 容易出现高方差；
2. 如果目标策略 π 和行为策略 b 的差异很大，那么这个方法收敛不太好。

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

n-step $tree - backup$ 算法示意图如下：

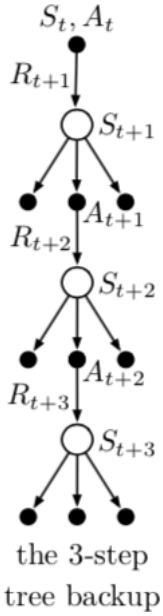


图 7.1: tree backup 算法示意图

一般来说，结点权重如下定义:

1. 第一级动作 a 对应的权重为 $\pi(a|S_{t+1})$;
2. 第二级中，总体的权重为 $\pi(A_{t+1}|S_{t+1})$, 其中未被选中的结点动作 a' 对应的权重为 $\pi(A_{t+1}|S_{t+1})\pi(a'|S_{t+2})$;
3. 同理第三级总权重为 $\pi(A_{t+1}|S_{t+1})\pi(A_{t+2}|S_{t+2})$, 其中未被选中的结点动作对应的权重为 $\pi(A_{t+1}|S_{t+1})\pi(A_{t+2}|S_{t+2})\pi(a''|S_{t+3})$;

假定在 target policy 中期望的价值函数:

$$V_t = \sum_a \pi(a|S_t)Q_{t-1}(S_t, a) \tag{7.14}$$

TD error 可以被定义成:

$$\delta_t = R_{t+1} + \gamma V_{t+1} - Q_{t-1}(S_t, A_t) \tag{7.15}$$

return 可以被定义成：

$$
\begin{aligned}
G_{t:t+1} &= R_{t+1} + \gamma V_{t+1} = Q_{t-1}(A_t, S_t) + \delta_t \\
G_{t:t+n} &= Q_{t-1}(A_t, S_t) + \sum_{k=t}^{m} in(t+n-1, T-1)\delta_k \prod_{t=t+1}^{k} \gamma\pi(A_i|S_i)
\end{aligned} \tag{7.16}
$$

n-step Sarsa 的 Q 值可以定义为：

$$
Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)] \tag{7.17}
$$

n-step Tree Backup for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

1. 随机地初始化 $Q(s, a)$;
2. 根据 Q 采用 ϵ-贪心算法初始化 π，或者根据实际需求初始化策略 π;
3. 参数: 步长参数 $\alpha \in (0, 1]$，$\epsilon > 0$，一个正整数 n;
4. 对于每次模拟，均进行如下操作：
   (a) 初始化 S_0 使其不在中断状态;
   (b) 选择策略 $A_0 \sim \pi(\cdot|S_0)$;
   (c) $Q_0 = Q(S_0, A_0)$;
   (d) $T \leftarrow \infty$;
   (e)

```
for t = 0, 1, 2, ··· && τ ≠ 0 do
    选择策略 A_0;
    观察获得下一个 reward R 和 S_{t+1};
    if S_{t+1} 是中断状态 then
        T ← t + 1;
        δ_t = R − Q_t;
    else
        δ_t = R + γ Σ_a π(a|S_{t+1}) Q(S_{t+1}, a) − Q_t;
        随机地选择一个动作 A_{t+1};
        Q_{t+1} = Q(S_{t+1}, A_{t+1});
```

```
        π_{t+1} = π(S_{t+1}, A_{t+1});
    end if
    当前时序下标:τ ← t − n + 1;
    if τ ≥ 0 then
        E ← 1;
        G ← Q_τ;
        for k = τ, ··· , min(τ + n − 1, T − 1) do;
            G ← G + Eδ_k;
            E ← γE_{π_{k+1}};
        end for
        Q(S_τ, A_τ) ← Q(S_τ, A_τ) + α[G − Q(S_τ, A_τ)];
        if 策略 π 已经被学习到 then
            确保 π(·|S_τ) 是关于 Q 的 ϵ-贪心法;
        end if
    end if
end for
```

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

为了比 ϵ-贪心算法进一步增加随机性，我们可以考虑探索 (exploration) 和期望 (expectation) 之间的连续变化，定义采样程度 $\sigma_t \in [0,1]$:

$$\sigma_t = \begin{cases} 1, & \text{只进行采样} \\ 0, & \text{不采样，输出期望} \end{cases} \tag{7.18}$$

单纯的采样造成的 $TD\ error$:

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n-1,T-1)} \gamma^{k-t}[R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k)] \tag{7.19}$$

我们把采样和期望的 TD error 通过 σ_t 进行概括：

$$\begin{aligned}\sigma_t &= R_{t+1} + \gamma[\sigma_{t+1}Q_t(S_{t+1}, A_{t+1}) + (1-\sigma_{t+1})\overline{Q}_{t+1}] - Q_{t-1}(S_t, A_t)\\ \overline{Q}_t &= \sum_a \pi(a|S_t)Q_{t-1}(S_t, a)\end{aligned} \tag{7.20}$$

所以我们可以定义 n-step $Q(\sigma)$ 的 return 了：

$$\begin{aligned}G_{t:t+1} &= \sigma_t + Q_{t-1}(S_t, A_t),\\ G_{t:t+n} &= Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n-1,T-1)} \sigma_k \prod_{i=t+1}^{k} \gamma[(1-\sigma_i)\pi(A_i|S_i) + \sigma_i]\end{aligned} \tag{7.21}$$

同时 importance sampling ratio 可以定义为：

$$\rho_{t:t+n} = \prod_{k=t}^{\min(t+n-1,T-1)} \left(\sigma_k \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} + 1 - \sigma_k\right) \tag{7.22}$$

算法如下所示：

Off-policy n-step $Q(\sigma)$ for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

1. **输入**: 一个随机的策略 b，只需保证 $b(a|s) > 0$ 即可；
2. 随机地初始化 $Q(s, a)$;
3. 根据 Q 采用 ϵ-贪心算法初始化 π，或者根据实际需求初始化策略 π;
4. 参数: 步长参数 $\alpha \in (0, 1]$，$\epsilon > 0$，一个正整数 n;
5. 对于每次模拟，均进行如下操作：
6. (a) 初始化 S_0 使其不在中断状态；
   (b) 选择策略 $A_0 \sim b(\cdot|S_0)$;
   (c) $Q_0 = Q(S_0, A_0)$;
   (d) $T \leftarrow \infty$;

(e)

for $t = 0, 1, 2, \cdots$ && $\tau \neq T - 1$ **do**
  if $t < T$ **then**
    选择动作 A_t;
    观察获得下一个 reward R 和 S_{t+1};
    if S_{t+1} 是中断状态 **then**
      $T \leftarrow t + 1$;
      $\delta_t = R - Q_t$;
    else
      随机地选择一个动作 $A_{t+1} \sim b(\cdot|S_{t+1})$;
      随机选择一个 σ_t
      $Q_{t+1} = Q(S_{t+1}, A_{t+1})$;
      $\sigma_t = R + \gamma\sigma_{t+1}Q_{t+1} + \gamma(1 - \sigma_{t+1}) \sum_a \pi(a|S_{t+1})Q(S_{t+1}|a) - Q_t$;
      $\pi_{t+1} = \pi(S_{t+1}, A_{t+1})$;
      $\rho_{t+1} = \frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}$;
    end if
  end if
  当前时序下标:$\tau \leftarrow t - n + 1$;
  if $\tau \geq 0$ **then**
    $\rho \leftarrow 1$;
    $e \leftarrow 1$;
    $G \leftarrow Q_\tau$;
    for $k = \tau, \cdots, \min(\tau + n - 1, T - 1)$ **do**
      $G \leftarrow G + e\delta_k$;
      $e \leftarrow \gamma e[(1 - \sigma_{k+1})\pi_{k+1} + \sigma_{k+1}]$;
      $\rho \leftarrow \rho(1 - \sigma_k + \sigma_k\rho_k)$;
    end for
    $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho[G - Q(S_\tau, A_\tau)]$;
    if 策略 π 已经被学习到 **then**

确保 $\pi(\cdot|S_\tau)$ 是关于 Q 的 ϵ-贪心法；

end if

end if

end for

Chapter 8

Planning and Learning with Tabular Methods

8.1 Models and Planning

- distribution models: 所有可能性;
  - 优点: 可以生成所有可能的过程, 以及各个过程的概率分布.
  - 缺点: 难以构建, 需要对环境足够的先验知识, 并且工程量浩大, 一旦模型出现错误无法更改.
- sample models: 所有可能性中的抽样（蒙特卡洛）;
  - 优点: 可以通过 agent 与环境交互的经验构建模型, 构建模型的方式更为简单, 并且可以更改模型中的错误.
  - 缺点: 只能生成一种可能的过程, 并且在概率估计上可能出现偏差.
- State-space planning: 通过状态空间搜索一个最优的政策或一个目标的最佳路径;
- plan-space planning: 动作导致状态到状态之间的转化，并且 V 值通过状态来进行计算;

Random-sample one-step tabular Q-planning

无限重复如下步骤：

1. 随机选择一个状态 S 和一个动作 A;
2. 将 S, A 发送给一个抽样模型，获得下一个状态 S' 和下一个 Reward(R);
3. $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',A) - Q(S,A)]$;

8.2 Dyna: Integrating Planning, Acting, and Learning

真实经验 (real experience) 的作用:

- 提升模型的性能 (model-learning)；
- 提升价值函数或策略 (direct reinforcement learning)；

它们的关系如下：

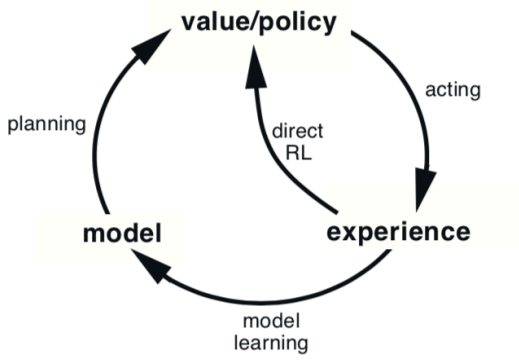


Figure 8.1: Relationships among learning, planning, and acting.

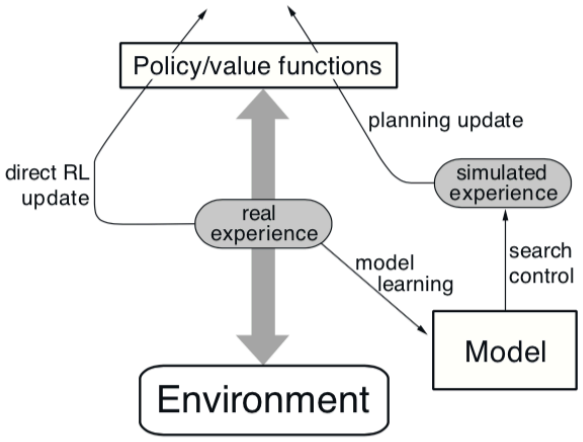


Figure 8.2: The general Dyna Architecture. Real experience, passing back and forth between the environment and the policy, affects policy and value functions in much the same way as does simulated experience generated by the model of the environment.

Tabular Dyna-Q

1. 初始化 $Q(s,a)$ 和 $Model(s,a)$;
2. 一直循环:
   (a) $S \leftarrow$ 当前非中断状态;
   (b) $A \leftarrow \epsilon - greedy(S,Q)$;
   (c) 执行动作 A，观察由此产生的 R, S';
   (d) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',A) - Q(S,A)]$;
   (e) $Model(S,A) \leftarrow R, S'$;
   (f) 以下步骤重复 n 次:
      i. $S \leftarrow$ 先前随机观测到的状态;
      ii. $A \leftarrow$ 在 S 状态下采用的 A;
      iii. $R, S' \leftarrow Model(S,A)$;
      iv. $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',A) - Q(S,A)]$;

8.3 When the Model Is Wrong

如果模型存在误差，那么计算出来的可能就不是最优策略了。

一般来说，Dyna-Q+ 相比于 Dyna-Q 来说具有一定的启发搜索能力，对于时间 τ 内未被尝试的搜索状态转移，其回报是 $r + \kappa\sqrt{\tau}$.

8.4 Prioritized Sweeping

如果我们采用均匀采样的话，会有许多价值函数的取值是 0，那么过多的 0 会导致 backup 操作无效而被浪费。我们从目标状态向前 (backward) 搜索 (回溯) 可能会是一个有效的策略. 我们希望该方法能够用于生成奖励函数，对于目标状态，仅仅是其中的一个特例而已。

prioritized sweeping 的思想是：在随机的环境中，变量状态转移的概率也对变化量的大小产生影响，可以根据 backup 的优先级进行排序，首先进行优先级别高的进行 backup：

Prioritized sweeping for a deterministic environment

1. 初始化 $Q(s,a)$,$Model(s,a)$，一个空队列 $PQueue$；
2. 一直重复:
   (a) $S \leftarrow$ 当前的非中断状态;
   (b) $A \leftarrow policy(S,Q)$;
   (c) 执行动作 A，观察由此产生的 R, S';
   (d) $Model(S,A) \leftarrow R, S'$;
   (e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$;
   (f) 如果 $P > \theta$，以优先级 P 将 S, A 插入 $PQueue$;
   (g) 如果 $PQueue$ 不为空，那么重复下面步骤 n 次：
      i. $S, A \leftarrow first(PQueue)$;
      ii. $R, S' \leftarrow Model(S,A)$;
      iii. $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',A) - Q(S,A)]$;

iv. 对于所有的 $\overline{S}, \overline{A}$ 预测值为 S 的，重复如下步骤:

A. $\overline{R} \leftarrow \overline{S}, \overline{A}, S$ 的预测回报;

B. $P \leftarrow |\overline{R} + \gamma \max_a Q(S, a) - Q(\overline{S}, \overline{A})|$.

C. 如果 $P > \theta$，以优先级 P 将 $\overline{S}, \overline{A}$ 插入 $PQueue$;

8.5 Full vs. Sample Backups

Full backup 没有抽样误差，对全局的估计将会更好，但是开销很大；在实际中用处不大，一般都是采用 sampling backups。

full backup 的动作价值函数可以写为:

$$Q(s, a) \leftarrow \sum_{s', r} \hat{p}(s', r|s, a) \left[r + \gamma \max_{a'} Q(s', a') \right] \tag{8.1}$$

其中 $\hat{p}(s', r|s, a)$ 是一个动态的估计模型，示意图如下:

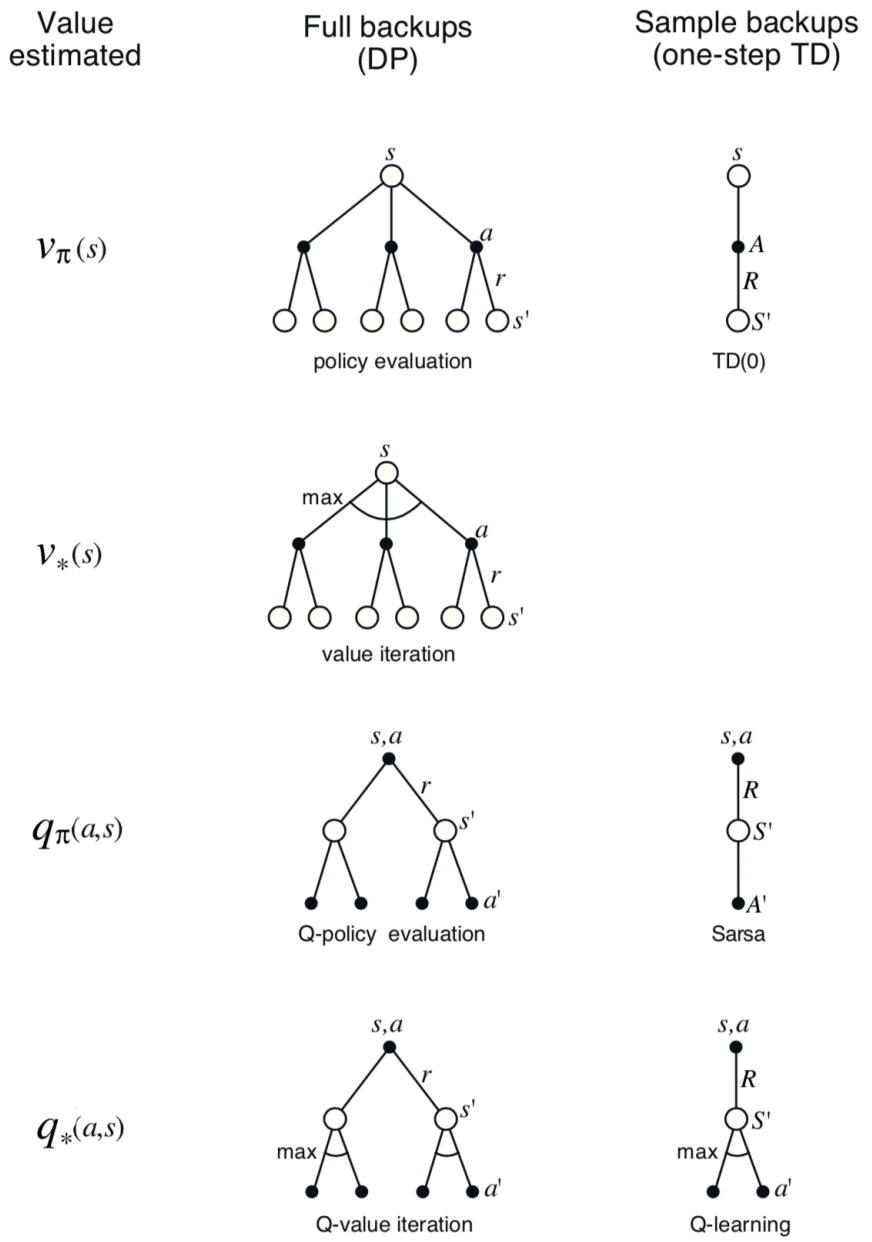


Figure 8.9: The one-step backups.

在这种条件下 Q-learning 的更新公式如下:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(S', a') - Q(s,a) \right] \tag{8.2}$$

这两种情况的不同的是环境的随机性，特别是给定一个状态和动作，许多可能情况都有许多可能性。如果有足够多的时间来进行一个 full backup，由于没有抽样误差，那么它的效果肯定会优于 sampling backups。

8.6 Trajectory Sampling

- The classical approach from dynamic programming: 通过对整个状态空间进行扫描，备份每一个状态-动作对。
- The second approach is to sample from the state or state–action space according to some distribution: 相对于第一种方式，第二种采用的是抽样代替全部扫描，这样的话可以加快扫描速度，避免一些极低概率的情况。

8.7 Real-time Dynamic Programming, RTDP

RTDP 是一个异步 (asynchronous) 的 DP 算法，在 RTDP 中，备份的顺序访问真实或者模拟的轨迹的状态顺序决定的。相比于普通 DP，它可以跳过一些不需要的状态，从而加快处理的速度。

RTDP 只处理与目标状态相关的状态，而忽略其余不相关的状态。

8.8 Planning at Decision Time

- background planning: 在当前状态 S_t，选择一个动作前。planning 需要对许多状态都选择动作。在这种情况下，planning 不集中在当前的状态了。
- decision-time planning: 每次遇到新的状态 S_t 就用 planning 来完成它, 不同于前面的 planning, 这在里，它将会确定一个特别方向。

8.9 Heuristic Search

启发式搜索 (Heuristic Search) 中，每个遇到的状态，可以将其认为是一个存在着许多可能性的一棵树。近似值函数 (the approximate value function) 可以认为是叶结点，然后以此回溯到根结点的当前状态。回溯完了选择最优结点即可。

不止一步 (one-step) 的深入搜索可以获得更好的动作选择，如果是一个好的模型和一个不完善的动作-价值函数，深入搜索可以获得更加好的策略。若进化到蒙特卡洛方法，那么这个不完善的动作-价值函数的误差将会被消除。如果搜索的深度为 k 而使得 γ^k 非常小，那么得到的策略也是接近最优的。但是越深的搜索会带来更大的计算开销。

一种做法是找出 backup 时最明显的部分，启发式搜索的有效性往往时集中在它的搜索树能够很快地得到当前的状态。如下图所示：

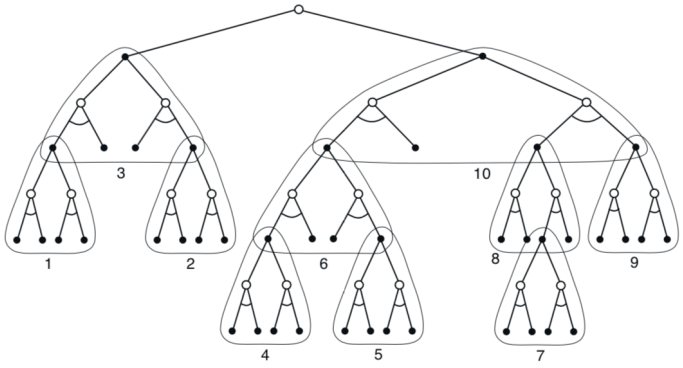


Figure 8.12: The deep backups of heuristic search can be implemented as a sequence of one-step backups (shown here outlined). The ordering shown is for a selective depth-first search.

8.10 Rollout Algorithms

Rollout 算法是一个基于蒙特卡洛的时间规划算法，适用于所有从当前环境状态开始的轨迹。是通过平均值来估计动作-价值函数。最优也是一个选择最大化价值函数的过程。

不同于蒙特卡洛方法，rollout 仅仅是针对当前的状态和给定的策略 (称

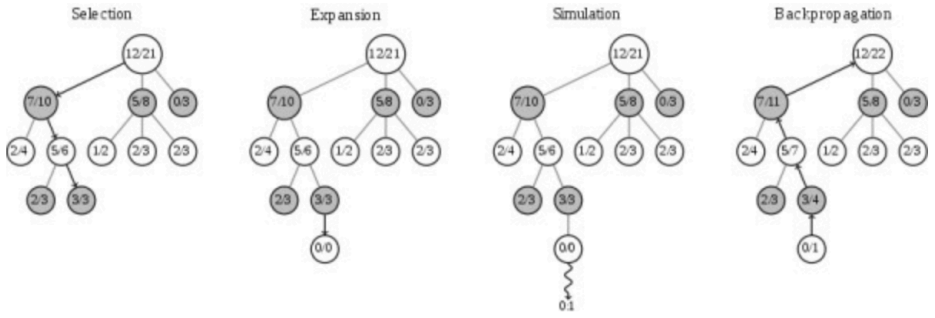
为起始策略)，往后估计蒙特卡洛值，而不是从头开始。这样可以使得计算变得比较简单，不需要对每个动作状态对产生采样，不需要在状态空间上产生近似函数。

8.11 Monte Carlo Tree Search

蒙特卡洛树搜索与蒙特卡洛方法相比：

- 蒙特卡洛方法得到的 reward 永远是后续步骤的 reward 均值，不会进步。在围棋游戏中存在偏差 (Bias) 和方差 (Variance).
- 蒙特卡洛树搜索可以向最优解进行收敛，在围棋游戏中不存在偏差，只有方差；但是对于复杂的局面，它仍有可能长期陷入陷阱，直到很久之后才能收敛得到正确答案。

以 AlphaGo 为例：



图中的数字代表：

$$\text{黑棋胜利次数/这个结点被访问次数} \tag{8.3}$$

图中的根结点 (12/21) 表示共模拟了 21 次，其中黑棋胜利了 12 次。下面介绍蒙特卡洛树搜索算法：

蒙特卡洛树搜索算法 (循环很多次)

1. 选择 (Selection): 从根结点往下走，在下层结点中按照如下公式选择一个结点：

$$S_{SelectBlack} = \arg\max \left(x_{child} + C \cdot \sqrt{\frac{\log(N_{parent})}{N_{child}}} \right) \tag{8.4}$$

其中 x_{child} 是子结点的胜率估计，N_{parent}, N_{child} 分别代表父结点和子结点的访问次数，而 C 是一个常数，C 越大就越偏向于广度搜索，C 越小就越偏向于深度搜索。由这个公式我们可以选择黑棋的最优结点。当到白棋走时，选择黑棋最不利的走法，即黑棋胜率最低的结点。

$$S_{SelectWlack} = \arg\min \left(x_{child} + C \cdot \sqrt{\frac{\log(N_{parent})}{N_{child}}} \right) \tag{8.5}$$

不断往下走，直到来到一个未拓展的结点 (叶结点)，如图上的 (3/3) 结点。

2. 扩展 (Expansion): 给 (3/3) 结点加入一个 (0/0) 的叶结点，即一种没有试过的走法。

3. 模拟 (Simluation): 通过这个新加入的结点，采用快速走子策略 (Rollout Policy) 走到底，得到一个胜负结果。

4. 回溯 (Backpropagation): 把胜负结果加到该叶结点，并加到该叶结点的所有父结点，如上图中得到的模拟结果是 (0/1), 则该结点的所有父结点都要加上 (0/1)。

Chapter 9

On-policy Prediction with Approximation

9.1 Value-function Approximation

明白:$s \mapsto g$,

- s:the state backed up;
- g:the backed-up value;

在不同的算法中:

- the Monte Carlo backup: $S_t \mapsto G_t$;
- the TD(0) backup: $S_t \mapsto R_{t+1} + \gamma\hat{v}(S_{t+1}, w_t)$;
- the n-step TD backup: $S_t \mapsto G_{t:t+n}$;
- the DP policy-evaluation backup: $s \mapsto \mathcal{E}_\pi[R_{t+1} + \gamma\hat{v}(S_{t+1}, w_t)|S_t = s]$;

9.2 The Prediction Objective (MSVE)

假设 $\mu(s) \geq 0$ 表示对于状态 s 的错误的重视程度，$\hat{v}(s, w)$ 表示近似值函数，$v_\pi(s)$ 表示真实值函数，那么均方值误差 (Mean Squared Value Error,

MSVE) 就可以写作：

$$MSVE(w) = \sum_{s \in \mathbb{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, w) \right]^2 \tag{9.1}$$

用上式的平方根来描述近似值与真实值之间的误差，这种方式叫做同策略分布 (on-policy distribution)。

9.3 Stochastic-gradient and Semi-gradient Methods

定义权重向量: $\mathbf{w} \doteq (w_1, w_2, \cdots, w_d)^T$; 用 $\mathbf{w}_t$ 代表再第 t 步的权重向量。近似值逼近的目的是通过有限的实例逼近所有的状态。

一个好的策略是尽量减少观察到的实例中的错误,随机梯度下降 (Stochastic gradient descent, (SGD)) 通过小幅度地调整权重向量，使其向最小化误差方向前进。

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha \nabla \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \right]^2 \\ &= \mathbf{w}_t + \alpha \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t) \end{aligned} \tag{9.2}$$

其中权重向量的偏导向量可以表示为：

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \cdots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right)^T \tag{9.3}$$

假设第 t 次抽样的样本是 U_t，我们可以用 U_t 代替 $\hat{v}(S_t, \mathbf{w}_t)$，并用下面的公式更新权重向量:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t) \tag{9.4}$$

该算法可以写为：

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

1. 输入：需要评估的策略 π;
2. 输入：一个误差函数 $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$;
3. 以适当的方式初始化价值权重向量 $\mathbf{W}$（如：$\mathbf{w} = 0$）;
4. 一直重复如下步骤
   (a) 采用策略 π 生成一条轨迹 $\langle S_0, A_0, R_1, S_1, A_1, \cdots, R_T, S_T$;
   (b) 对于轨迹中的每一步 $t \in [0, T-1]$:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[G_t - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w})$$

如果我们想用到的并不是轨迹的全部，而是部分，那么将转化为 TD(n) 算法，这样做的目的是只考虑权重向量改变在估计中的影响，而不是对目标的影响。这个算法可以描述成：

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

1. 输入：需要评估的策略 π;
2. 输入：一个误差函数 $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$ 且 $\hat{v}(terminal, \cdot) = 0$;
3. 以适当的方式初始化价值权重向量 $\mathbf{W}$（如：$\mathbf{w} = 0$）;
4. 在每条轨迹中进行如下步骤
   (a) 初始化 S;
   (b) 对于每条轨迹中的每一步，重复下面的步骤，直到 S 到达中断状态;
      i. 从策略 $\pi(\cdot|S)$ 中选择动作 A;
      ii. 通过动作 A，获得 R, S';
      iii. $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})]\nabla\hat{v}(S, \mathbf{w})$;
      iv. $S \leftarrow S'$;

9.4 Linear Methods

将近似值函数用真实值 (real-valued) 向量 $\mathbf{x}(s)$ 和权重向量 $\mathbf{w}$ 线性表示出来：

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s) = \sum_{i=1}^{d} w_i x_i(s) \tag{9.5}$$

在这种情况下，近似值函数相对于 $\mathbf{w}$ 的梯度可以表示为:

$$\nabla \hat{v}(s, \mathbf{w}) = \mathbf{x}(s) \tag{9.6}$$

在半 (semi) 梯度 TD(0) 算法下，每一步指向的不是全局最优，而是当前最优，更新公式可以写为：

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha(R_{t+1} + \gamma \mathbf{w}_t^T \mathbf{x}_{t+1} - \mathbf{w}_t^T \mathbf{x}_t)\mathbf{x}_t \\ &= \mathbf{w}_t + \alpha \left(R_{t+1}\mathbf{x}_t - \mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^T \mathbf{w}_t\right) \end{aligned} \tag{9.7}$$

当到达稳定状态时，$\mathbf{w}_{t+1}$ 可以由 $\mathbf{w}_t$ 推导出来：

$$\begin{aligned} &\mathbb{E} = [\mathbf{w}_{t+1} | \mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t) \\ &\mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t] \in \mathbb{R}^d \\ &\mathbf{A} \doteq \mathbb{E}[\mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^T] \in \mathbb{R}^d \times \mathbb{R}^d \end{aligned} \tag{9.8}$$

如果系统收敛，那么它的权重向量将收敛到 $\mathbf{w}_{TD}$:

$$\begin{aligned} &\mathbf{b} - \mathbf{A}\mathbf{w}_{TD} = 0 \\ &\Rightarrow \mathbf{b} = \mathbf{A}\mathbf{w}_{TD} \\ &\Rightarrow \mathbf{w}_{TD} = \mathbf{A}^{-1}\mathbf{b} \end{aligned} \tag{9.9}$$

这时候称 $\mathbf{w}_{TD}$TD 不动点 (fixedpoint)。在这个点，有界的均方误差 (MSVE) 达到最小的可能误差:

$$\text{MSVE}(\mathbf{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\mathbf{w}} \text{MSVE}(\mathbf{w}) \tag{9.10}$$

将其拓展到 semi-gradient TD(n) 算法，则：

$$\begin{aligned} &\mathbf{w}_{t+n} = \mathbf{w}_{t+n-1} + \alpha[G_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1})]\nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}) \\ &G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}) \end{aligned} \tag{9.11}$$

算法可以写为：

n-step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

1. 输入：需要评估的策略 π;
2. 输入：一个误差函数 $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$ 且 $\hat{v}(terminal, \cdot) = 0$;
3. 参数：步长 $\alpha \in (0, 1]$，自然数 n;
4. 随机初始化价值权重向量 $\mathbf{w}$, 如 $(\mathbf{w} = 0)$;
5. 对于每条轨迹，重复下面步骤：
   (a) 初始化 S_0，使其不处于中断状态；
   (b) $T \leftarrow \infty$;
   (c)

```
for t = 0, 1, 2, ··· &&τ ≠ T − 1 do
    if t < T then
        根据 π(·|S_t) 选择动作 A;
        根据 A 获得 R_{t+1}, S_{t+1};
        if S_{t+1} 是中断的 then
            T ← t + 1;
        end if
    end if
    τ ← t − n + 1;
    if τ ≥ 0 then
        G ← Σ_{i=τ+1}^{min(τ+n,T)} γ^{i−τ−1} R_i;
        if τ + n < T then
            G ← G + γ^n v̂(S_{τ+n}, w);
        end if
        w ← w + α[G − v̂(S_τ, w)]∇v̂(S_τ, w);
    end if
end for
```

9.5 Feature Construction for Linear Methods

9.5.1 Polynomials(多项式)

在多维连续状态空间 (multi-dimensional continuous state spaces) 中，函数值逼近相似于插值与回归 (interpolation and regression).

该思想可以描述为:

对于 d 维的变量，每个状态 s 可以写为一个 d 维的向量 $(s_1, s_2, \cdots, s_d)^T$，那么每个多项式函数 x_i 可以写为:

$$x_i(s) = \prod_{j=1}^{d} s_j^{c_{i,j}} \tag{9.12}$$

9.5.2 Fourier Basis(傅立叶基)

在函数值逼近的过程中，如果需要逼近的目标函数值未知，那么比较好的方式就是使用傅立叶基的函数了。

对于 d 维在一个单位超立方体 (unit hypercube) 中的状态向量 $(s_1, s_2, \cdots, s_d)^T, s \in [0, 1]$, 采用傅立叶基改写 x_i:

$$x_i(s) = \cos(\pi c^i \cdot s) \tag{9.13}$$

对应的步长参数:

$$\alpha_i = \frac{\alpha}{\sqrt{\sum_{k=1}^{d} (c_k^i)^2}} \tag{9.14}$$

9.5.3 Coarse Coding

Coarse coding 通常用来处理有许多重叠特征的状态。

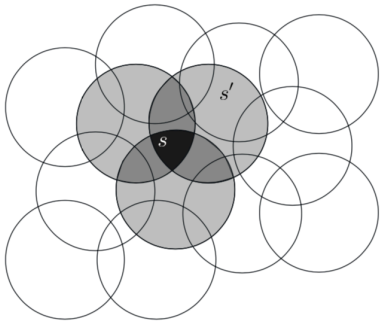


Figure 9.6: Coarse coding. Generalization from state s to state s' depends on the number of their features whose receptive fields (in this case, circles) overlap. These states have one feature in common, so there will be slight generalization between them.

如果状态是存在一个圆之内，那么对应的特征值为 1，我们认为它是存在的，否则特征值为 0，认为他是不存在的。这样我们就会获得一个二进制的特征，对于一个给定的状态，它的特征可以用所有的圆进行表示，这种编码方式就是 Coarse Coding。

9.5.4 Tile Coding

Tile Coding 是多维连续状态空间中更加流畅和计算效率高的 Coarse coding。在 tile 编码中，特征的接收域被划分到输入空间的一个被称为 tiling 的小分区。每个分区的元素称为 tile。如下图所示：

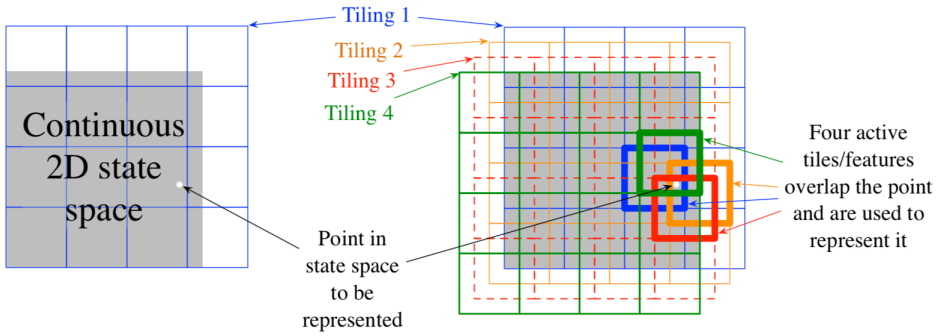


Figure 9.9: Multiple, overlapping grid-tilings on a limited two-dimensional space. These tilings are offset from one another by a uniform amount in each dimension.

假设 tiling 的数目为 m，那么步长就可以设为 $\alpha = \frac{1}{m}$。由于编码最后只有 0 和 1，类比于 one-hot。

9.5.5 Radial Basis Functions(径向基函数)

在一个典型的径向基函数中，$x_i(s)$ 具有一定的高斯效应，中心状态设为 c_i，特征宽度为 σ_i，那么:

$$x_i(s) = \exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right) \tag{9.15}$$

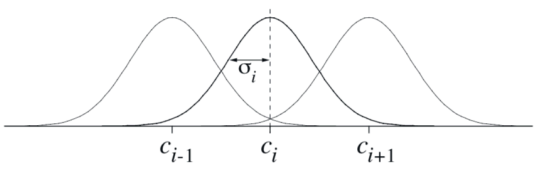


Figure 9.13: One-dimensional radial basis functions.

9.6 Nonlinear Function Approximation: Artificial Neural Networks

这节主要是介绍神经网络的，没有什么干货。

9.7 Least-Squares TD, LSTD

在 9.4 节已经算出了 fixedpoint 为:$\mathbf{w}_{TD} = \mathbf{A}^{-1}\mathbf{b}$. 前面的方法是需要迭代 (iteratively) 的, 在本节介绍的 LSTD 就可以做到不需要迭代。

假设:

$$\begin{aligned}\hat{\mathbf{A}}_t &\doteq \sum_{k=0}^{t} \mathbf{x}_k(\mathbf{x}_k - \gamma\mathbf{x}_{k+1})^T + \varepsilon\mathbf{I} \\ \hat{\mathbf{b}}_t &\doteq \sum_{k=0}^{t} R_{t+1}\mathbf{x}_k\end{aligned} \tag{9.16}$$

其中 $\varepsilon\mathbf{I} > 0$ 只是为了保证 $\hat{\mathbf{A}}_t$ 是可逆的。故 TD fixedpoint 可以写为：

$$\mathbf{w}_{t+1} \doteq \hat{\mathbf{A}}_t^{-1}\hat{\mathbf{b}}_t \tag{9.17}$$

LSTD 的空间复杂度为 $O(d^2)$ 而时间复杂度 $O(d^2)$，每一步计算的空间复杂度和时间复杂度均为 $O(d)$。

LSTD 的另一个难题是如何求 $\hat{\mathbf{A}}_t$ 的逆矩阵 $\hat{\mathbf{A}}_t^{-1}$，一般采用下面的公式进行求解：

$$\begin{aligned}\hat{\mathbf{A}}_t^{-1} &= \left(\hat{\mathbf{A}}_{t-1} + \mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^T\right)^{-1} \\ &= \hat{\mathbf{A}}_{t-1}^{-1} - \frac{\hat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^T\hat{\mathbf{A}}_{t-1}^{-1}}{1 + (\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^T\hat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_t}\end{aligned} \tag{9.18}$$

其中 $\hat{\mathbf{A}}_{-1} = \varepsilon\mathbf{I}$，我们可以存储 $\widehat{\mathbf{A}^{-1}}_t$ 作为中间变量，从而得到如下算法：

LSTD for estimating $\hat{v} \approx v_\pi$ ($O(d^2)$ version)

1. 输入：特征向量 $\mathbf{x}(s) \in \mathbb{R}^d, \forall s \in \mathcal{S}, \mathbf{x}(terminal) \doteq 0$；
2. $\widehat{\mathbf{A}^{-1}} \leftarrow \varepsilon^{-1}\mathbf{I}$；
3. $\hat{\mathbf{b}} \leftarrow 0$；
4. 对于每条轨迹，重复下面步骤：
   (a) 初始化 S 和相应的 $\mathbf{x}$；
   (b) 对于轨迹的每一步，重复如下步骤，直到 S 到达中断状态：
      i. 从策略 $\pi(\cdot|S)$ 中选择动作 A；
      ii. 根据动作 A, 获得 R, S' 及相应的 $\mathbf{x}'$；
      iii. $\mathbf{v} \leftarrow \widehat{\mathbf{A}^{-1}}^T(\mathbf{x} - \gamma\mathbf{x}')$；
      iv. $\widehat{\mathbf{A}^{-1}} \leftarrow \widehat{\mathbf{A}^{-1}} - \frac{(\widehat{\mathbf{A}^{-1}}\mathbf{x})\mathbf{v}^T}{1+\mathbf{v}^T\mathbf{x}}$；
      v. $\hat{\mathbf{b}} \leftarrow \hat{\mathbf{b}} + R\mathbf{x}$；
      vi. $\theta \leftarrow \widehat{\mathbf{A}^{-1}}\hat{\mathbf{b}}$；
      vii. $S \leftarrow S'; \mathbf{x} \leftarrow \mathbf{x}'$；

9.8 Memory-based Function Approximation

Memory-based 将需要训练的样本进行存储，不进行任何操作，当需要进行逼近时，再从头开始计算，是一种懒惰学习 (lazy-learning)。

9.9 Kernel-based Function Approximation

核回归技巧应用到函数值逼近之中，其本质是计算一个所有目标的核平均权重。假设 D 是已经存储的例子，$g(s')$ 表示状态 s' 的目标。那么逼近函数 $\hat{v}(s, D)$ 可以写为:

$$\hat{v}(s, D) = \sum_{s' \in D} k(s, s') g(s') \tag{9.19}$$

9.10 Looking Deeper at On-policy Learning: Interest and Emphasis(重要性)

与前面的算法不同，这节的算法给每个状态加上了权重的概念。用 $I_t \in [0, 1]$ 表示在 t 时间时我们感兴趣的程度，其次，我们引入一个随机的重要性 (emphasis) 指数 $M_t \in [0, 1]$。那么价值权重向量可以写作:

$$\begin{aligned} \mathbf{w}_{t+n} &\doteq \mathbf{w}_{t+n-1} + \alpha M_t [G_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1}) \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}) \\ M_t &= I_t + \gamma^n M_{t-n} \end{aligned} \tag{9.20}$$

当 $M_t = 0$ 时，表示该状态没有参考性，有 $G_{t:t+n} = G_t$。

9.11 Deep Q-Network, DQN

首先我们要明白在普通的 Q-learning 的算法，$Q(S, A)$ 的更新公式如下:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \lambda \max_a Q(S_{t+1}, a) - Q(S_t, A_t))] \tag{9.21}$$

按照上面的公式，说明对于所有的状态 $s \in S$, 所有的动作 $a \in A$，我们可以建立一张表来存储 $Q(s, a)$ 的值。但这是不现实的，因为需要的存储空间太大了 ($O(\text{count}(s) \times \text{count}(a))$)，形成了维数灾难.

为了避免维数灾难，我们采用值函数近似 (Value Function Approximation) 的方式，生成一个函数 $f(s,a,w)$ 来逼近 $Q(s,a)$：

$$f(s,a,w) \approx Q(s,a) \tag{9.22}$$

其中 w 代表 f 的所有参数的集合。

现在的问题是如何求出 $f(s,a)$，由于游戏基本上的输入量 (基本上都是输入一帧画面) 都是远远大于输出量 (一些离散的控制数值)，所以比较好的求解方式是通过神经网络 (neutral network) 进行逼近。

到这里 DQN 算法就很简单了，我们只需要通过神经网络 (CNN, BPNN 等) 求出 $f(s,a,w)$，再由 $f(s,a,w)$ 逼近 $Q(s,a)$，这样就可以避免需要存储许多 Q 值了。

神经网络的原理大致都类似：最小化损失函数 $\mathcal{L}(s)$，常见的做法是最小化均方误差 (MVSE):

$$\begin{aligned} w &= \arg\min \mathcal{L}(w) \\ &= \arg\min \left(\mathbb{E}[(r + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w))^2] \right) \end{aligned} \tag{9.23}$$

其中 s', a' 分别代表下一个动作和状态。通过训练神经网络到达收敛之后，我们可以得到 w 从而确定 $f(s,a,w)$，之后即可计算出在给定 (s,a) 的情况下，对应的 $Q(s,a)$ 的大小了。

Chapter 10

On-policy Control with Approximation

10.1 Episodic Semi-gradient Control

一般来说，采用梯度下降算法更新动作-价值预测的公式：

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[U_t - \hat{q}(S_t, A_t, \mathbf{w}_t)\right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \tag{10.1}$$

其中 U_t 是任意的 $q_\pi(S_t, A_t)$. 如果，我们采用上式更新 one-step Sarsa 的话，那么得到的就是 Episodic Semi-gradient one-step Sarsa 算法：

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)\right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \tag{10.2}$$

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

1. 输入：一个差异 (differentiable) 的函数:$\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$;
2. 初始化：随机初始化价值函数的权重 $\mathbf{w} \in \mathbb{R}^d$ ，如 $(\mathbf{w} = 0)$;
3. 对于每条轨迹，重复如下步骤:
   (a) 采用 ϵ-贪心算法初始化 S, A;
   (b) 对于一条轨迹的每一步，重复如下步骤:

(c) 1: 执行动作 A，获取 R, S';
2: **if** S' 是中断状态 **then**
3: $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$;
4: **continue**;
5: **end if**
6: 选择一个动作 A'，采用 ϵ-贪心算法，以便获得 $\hat{q}(S', \cdot, \mathbf{w})$;
7: $\mathbf{w} \leftarrow \mathbf{w} + \alpha\left[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\right]\nabla\hat{q}(S, A, \mathbf{w})$;
8: $S \leftarrow S'$;
9: $A \leftarrow A'$;

10.2 n-step Semi-gradient Sarsa

上节讲的是单步的 Sarsa 算法，我们可以用一个 n 步回报值来作为 Sarsa 的更新算法：

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}) \quad (10.3)$$

故，更新公式可以写为：

$$\mathbf{w}_{t+n} = \mathbf{w}_{t+n-1} + \alpha\left[G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1})\right]\nabla\hat{q}(S_t, A_t, \mathbf{w}_{t+n-1}) \quad (10.4)$$

基于此，算法可以写为:

Episodic semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_*$, or $\hat{q} \approx q_\pi$

1. 输入：一个差异 (differentiable) 的函数:$\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$;
2. 初始化：随机初始化价值函数的权重 $\mathbf{w} \in \mathbb{R}^d$ ，如 $(\mathbf{w} = 0)$;
3. 参数：步长 $\alpha > 0$,$\epsilon > 0$,n;
4. 对每条轨迹，重复如下步骤：
5. (a) 初始化 S_0，并使其不处于中断状态;

(b) 选择一个初始动作 $A_0 \sim \pi(\cdot|S_0)$ 或者关于 (wrt)$\hat{q}(S_0, \cdot, \mathbf{w})$ 的 ϵ-贪心算法。

(c) $T \leftarrow \infty$;

(d)
1: **for** $t = 0, 1, 2, \cdots \&\&\tau \neq T - 1$ **do**
2: **if** $t < T$ **then**
3: 选择动作 A_t;
4: 执行动作 A_t，获得 R_{t+1} 和 S_{t+1};
5: **if** S_{t+1} 处于中断状态 **then**
6: $T \leftarrow t + 1$;
7: **else**
8: 选择一个动作 $A_{t+1} \sim \pi(\cdot|S_{t+1})$ 或者关于 (wrt) $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$ 的 ϵ-贪心算法。
9: **end if**
10: **end if**
11: $\tau \leftarrow t - n + 1$;
12: **if** $\tau \geq 0$ **then**
13: $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$;
14: **if** $\tau + n < T$ **then**
15: $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w})$;
16: **end if**
17: $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{q}(S_\tau, A_\tau, \mathbf{w})\right] \nabla \hat{q}(S_\tau, A_\tau, \mathbf{w})$;
18: **end if**
19: **end for**

10.3 Average Reward: A New Problem Setting for Continuing Tasks

由于折扣函数 γ 会对函数值逼近造成困难，所以我们要使用平均奖励 (average-reward) 来代替，设策略 π 的平均奖励为 $r(\pi)$:

$$\begin{aligned} r(\pi) &\doteq \lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[R_t|A_{0:t-1}\sim\pi] \\ &= \lim_{t\to\infty}\mathbb{E}[R_t|A_{0:t-1}\sim\pi] \\ &= \sum_s \mu_\pi(s)\sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)r \end{aligned} \tag{10.5}$$

其中 $\mu_\pi(s) = \lim_{t\to\infty} Pr\{S_t = s|A_{0:t-1}\sim\pi\}$ 是一个稳态分布 (steady-state distribution), 称为遍历 (ergodicity) 性的.

回报 G_t 就可以被定义为:

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots \tag{10.6}$$

称为差异回报 (differential return)。

在均值回报下，状态值函数和动作-价值函数可以写为:

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s)\sum_{r,s'} p(s',r|s,a)\left[r - r(\pi) + v_\pi(s')\right] \\ q_\pi(s,a) &= \sum_{r,s'} p(s',r|s,a)\left[r - r(\pi) + \sum_{a'}\pi(a'|s')q_\pi(s',a')\right] \\ v_*(s) &= \max_a \sum_{r,s'} p(s',r|s,a)\left[r - r(\pi) + v_*(s')\right] \\ q_*(s,a) &= \sum_{r,s'} p(s',r|s,a)\left[r - r(\pi) + \max_{a'} q_*(s',a')\right] \end{aligned} \tag{10.7}$$

TD-error 可以写作:

$$\begin{aligned} \delta_t &\doteq R_{t+1} - \overline{R}_{t+1} + \hat{v}(S_{t+1},\mathbf{w}_t) - \hat{v}(S_t,\mathbf{w}_t) \\ \delta_t &\doteq R_{t+1} - \overline{R}_{t+1} + \hat{q}(S_{t+1},A_{t+1},\mathbf{w}_t) - \hat{q}(S_t,A_t,\mathbf{w}_t) \end{aligned} \tag{10.8}$$

其中 $\overline{R}_t$ 是平均回报 $r(\pi)$ 在时间 t 时候的估计。那么采用平均回报的 semi-gradient Sarsa 算法的更新公式可以写作：

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \tag{10.9}$$

Differential semi-gradient Sarsa for estimating $\hat{q} \approx q_*$

1. 输入：一个差异 (differentiable) 的函数:$\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$;
2. 步长参数：$\alpha, \beta > 0$;
3. 随机地初始化状态值函数权重向量 $\mathbf{w} \in \mathbb{R}^d$，如 $\mathbf{w} = 0$;
4. 随机地初始化平均回报估计 $\overline{R}$，如 $\overline{R} = 0$;
5. 初始化 S, A;
6. 对于每一步，重复如下步骤：
7. (a) 执行动作 A，观察 R, S';
   (b) 为 $\hat{q}(S', \cdot, \mathbf{w})$ 通过例如 ϵ-贪心算法选择动作 A';
   (c) $\delta \leftarrow R - \overline{R} + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$;
   (d) $\overline{R} \leftarrow \overline{R} + \beta\delta$;
   (e) $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S, A, \mathbf{w})$;
   (f) $S \leftarrow S'$;
   (g) $A \leftarrow A'$;

10.4 Deprecating the Discounted Setting

在无限长的序列中，我们要采用极限的思想求平均奖励：

$$\mathcal{J}(\pi) = r(\pi) \times (1 + \gamma + \gamma^2 + \cdots + \gamma^n) \doteq \frac{r(\pi)}{1 - \gamma} \tag{10.10}$$

10.5 n-step Differential Semi-gradient Sarsa

n-step Sarsa 中，回报奖励可以写为:

$$G_{t:t+n} \doteq R_{t+1} - \overline{R}_{t+1} + \cdots + R_{t+n} - \overline{R}_{t+n} + \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}) \quad (10.11)$$

那么 n-step TD error 可以写为:

$$\delta_t \doteq G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}) \quad (10.12)$$

Differential semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_*$, or $\hat{q} \approx q_\pi$

1. 输入: 一个差异 (differentiable) 的函数:$\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$;
2. 步长参数: $\alpha, \beta > 0$，一个自然数 n;
3. 随机地初始化状态值函数权重向量 $\mathbf{w} \in \mathbb{R}^d$，如 $\mathbf{w} = 0$;
4. 随机地初始化平均回报估计 $\overline{R}$，如 $\overline{R} = 0$;
5. 初始化 S, A;
6. 1: **for** $t = 0, 1, 2, \cdots$ **do**
   2: 选择一个动作 A_t;
   3: 获取 R_{t+1}, S_{t+1};
   4: 选择一个动作 $A_{t+1} \sim \pi(\cdot|S_{t+1})$ 或者关于 (wrt) $\hat{q}(S_0, \cdot, \mathbf{w})$ 的 ϵ-贪心算法。
   5: $\tau \leftarrow t - n + 1$;
   6: **if** $\tau \geq 0$ **then**
   7: $\delta \leftarrow \sum_{i=\tau+1}^{\tau+n}(R_i - \overline{R}) + \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w}) - \hat{q}(S_\tau, A_\tau, \mathbf{w})$;
   8: $\overline{R} \leftarrow \overline{R} + \beta\delta$;
   9: $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S_\tau, A_\tau, \mathbf{w})$;
   10: **end if**
   11: **end for**

Chapter 11

Off-policy Methods with Approximation

11.1 Semi-gradient Methods

在函数值近似中，每步重要性采样比例公式可以写为：

$$\rho_t \doteq \rho_{t:t} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)} \tag{11.1}$$

比如说，semi-gradient off-policy TD(0) 算法的权重向量更新可以写为：

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\rho_t\delta_t\nabla\hat{v}(S_t, \mathbf{w}) \tag{11.2}$$

其中 δ_t 是采用平均回报的 TD error：

$$\begin{aligned} \delta_t &\doteq R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t), \text{or} \\ \delta_t &\doteq R_{t+1} - \overline{R}_t + \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \end{aligned} \tag{11.3}$$

对于动作-价值来说，上面的算法就变成了 one-step semi-gradient Expected Sarsa：

$$\begin{aligned} &\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\rho_t\delta_t\nabla\hat{q}(S_t, A_t, \mathbf{w}_t), \text{with} \\ &\delta_t \doteq R_{t+1} + \gamma\sum_a \pi(a|S_t+1)\hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t), \text{or} \\ &\delta_t \doteq R_{t+1} - \overline{R}_t + \sum_a \pi(a|S_{t+1})\hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \end{aligned} \tag{11.4}$$

同理，n-step semi-gradient Expected Sarsa 可以写为：

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha\rho_{t+1}\cdots\rho_{t+n-1}\left[G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1})\right]\nabla\hat{q}(S_t, A_t, \mathbf{w}_{t+n-1}),$$

with

$$\begin{aligned} G_{t:t+n} &\doteq R_{t+1} + \cdots + \gamma^{n-1}R_{t+n} + \gamma^n\hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}), \text{or} \\ G_{t:t+n} &\doteq R_{t+1} - \overline{R}_t + \cdots + R_{t+n} - \overline{R}_{t+n-1} + \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}) \end{aligned} \tag{11.5}$$

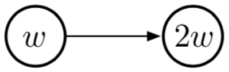
n-step semi-gradient tree-backup 算法可以写为：

$$\begin{aligned} &\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha[G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1})]\nabla\hat{q}(S_t, A_t, \mathbf{w}_{t+n-1}), \text{with} \\ &G_{t:t+n} \doteq \hat{q}(S_t, A_t, \mathbf{w}_{t-1}) + \sum_{k=t}^{t+n-1}\delta_k\prod_{i=t+1}^{k}\gamma\pi(A_i|S_i) \end{aligned} \tag{11.6}$$

11.2 Examples of Off-policy Divergence

在近似值逼近中会遇到一个困难：行为策略的分布和目标策略的分布不一致 (the distribution of updates does not match the on-policy distribution)。

在下图中，有两个状态，一个动作，从左边状态到右边状态的 reward 为 0：



两个状态进行转换时产生的 TD error:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \\ &= 0 + \gamma 2w_t - w_t \\ &= (2\gamma - 1)w_t \end{aligned} \tag{11.7}$$

off-policy semi-gradient TD(0) 更新公式：

$$\begin{aligned} w_{t+1} &= w_t + \alpha\rho_t\delta_t\nabla\hat{v}(S_t, w_t) \\ &= w_t + \alpha\cdot 1\cdot(2\gamma - 1)w_t\cdot 1 \\ &= (1 + \alpha(2\gamma - 1))w_t \end{aligned} \tag{11.8}$$

其中重要性采样比率 $\rho_t = 1$。

这个例子的关键是，对于一个转化 (行为策略)，它重复发生时 w 在目标策略上没有发生更新。发生这样的原因是由于行为策略可能采样时会选择目标策略可能永远不会选择的行为。这说明了目标策略和行为策略会造成差异。

为了减小这种差异，有两种途径：

- 采用重要性采样，来平衡行为策略之中的权重。
- 采用一个不依赖于样本的随机梯度下降策略。

11.3 The Deadly Triad

我们结合下面三个要素容易造成系统的不稳定与差距，所以我们称之为 (The Deadly Triad):

- **Function approximation**: 一种强大、可扩展的方法，使用大量的空间和计算资源来生成状态空间，（如线性函数近似、人工神经网络）；
- **Bootstrapping**: 更新包括现有估计的目标时完全依靠实际回报，和完全的回报。(如 DP 算法和 TD 算法)
- **Off-policy training**: 离策略指的是关于目标策略之外的另外的训练，如 DP 算法一样，扫描所有的状态空间并更新所有的状态，这个行为不需要遵循目标策略。

这个误差是无法避免的，至于解决方法到目前为止也没有太好的办法，只能是采用上节所提到的途径:

- 采用重要性采样，来平衡行为策略之中的权重。
- 采用一个不依赖于样本的随机梯度下降策略。

11.4 Linear Value-function Geometry

我们要注意的是，在绝大多数情况下，大多数价值函数并不符合任何的策略。对我们而言，更多的并不是需要通过函数值进行逼近，而是通过设计比状态量更少的参数 (More important for our purposes is that most are not representable by the function approximator, which by design has far fewer parameters than there are states.)??

我们给出三个状态 $\mathcal{S} = \{s_1, s_2, s_3\}$ 和两个参数 $\mathbf{w} = (w_1, w_2)^T$。在一个三维空间中，对于任意一组数 (x, y)，我们设 $w_1 = x, w_2 = y$ 那么可以得到下图：

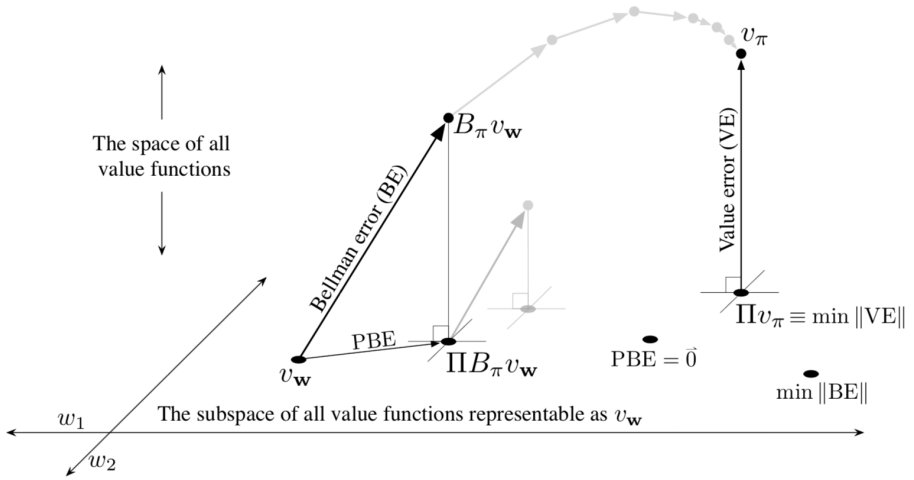


Figure 11.3: The geometry of linear value-function approximation. Shown as a plane is the subspace of all functions representable by the function approximator. The three-dimensional space above and below it is the much larger space of all value functions (functions from $\mathcal{S}$ to $\mathbb{R}$). The true value function v_π is in this larger space and projects down to its best approximation in the value error (VE) sense. The best approximators in the Bellman error (BE) and projected Bellman error (PBE) senses are different and are also shown in the lower right. (VE, BE, and PBE are all treated as the corresponding vectors in this figure.) The Bellman operator takes a value function in the plane to one outside, which can then be projected back. If you could iteratively apply the Bellman operator outside the space (shown in gray above) you would reach the true value function, as in conventional DP.

假设一个固定的额策略 π，假设它的真实状态值函数是 v_π。为了度量两个不同的价值函数 (目标策略和行为策略)，有 $v = v_1 - v_2$。我们可以用权

重值 $\mu : \mathcal{S} \to \mathbb{R}$ 权衡我们关心的不同状态的精确量化。综上，我们可以定义如下公式用来衡量状态值函数之间的差异：

$$\|v\|_\mu^2 \doteq \sum_{s \in \mathcal{S}} \mu(s) v(s)^2 \tag{11.9}$$

引入均方误差 (MSVE) 有:

$$\text{MSVE}(\mathbf{w}) = \|v_\mathbf{w} - v_\pi\|_\mu^2 \tag{11.10}$$

我们可以定义一个操作 Π 来表示最接近我们要求的状态值函数:

$$\begin{aligned} \Pi v &\doteq v_\mathbf{w} \\ &\text{where,} \\ \mathbf{w} &= \arg\min_\mathbf{w} \|v - v_\mathbf{w}\|_\mu^2 \end{aligned} \tag{11.11}$$

最接近真实值函数 v_π 是 Π 的投影，如下面所示：

The projection matrix

对于一个线性函数逼近，它的投影也是线性的，可以写为一个 $|\mathcal{S}| \times |\mathcal{S}|$ 的矩阵：

$$\Pi \doteq \mathbf{X} \left(\mathbf{X}^T D \mathbf{X} \right)^{-1} \mathbf{X}^T D \tag{11.12}$$

其中:

$$D \doteq \begin{bmatrix} \mu(1) & & & 0 \\ & \mu(2) & & \\ & & \ddots & \\ 0 & & & \mu(|\mathcal{S}|) \end{bmatrix} \tag{11.13}$$

$$\mathbf{X} \doteq \begin{bmatrix} -\mathbf{x}(1)^T - \\ -\mathbf{x}(2)^T - \\ \vdots \\ -\mathbf{x}(|\mathcal{S}|)^T - \end{bmatrix} \tag{11.14}$$

若上述存在广义逆矩阵，那么标准向量可以写为:

$$\|v\|_\mu^2 = v^T D v \tag{11.15}$$

线性逼近价值函数可以重写为:

$$v_{\mathbf{w}} = \mathbf{X}\mathbf{w} \tag{11.16}$$

在 TD 算法中，v_π 可以写为:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \quad \forall s \in \mathcal{S} \tag{11.17}$$

我们可以定义出 Bellman error:

$$\begin{aligned} \overline{\delta}_{\mathbf{w}}(s) &\doteq \left(\sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_{\mathbf{w}}(s')] \right) - v_{\mathbf{w}}(s) \\ &= \mathbb{E}[R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)|S_t = s, A_t \sim \pi] \end{aligned} \tag{11.18}$$

这个公式说明了 Bellman error 是 TD error 的期望。引入均方 Bellman 误差 (Mean Squared Bellman Error, MSBE) 和 Bellman 误差向量, 有:

$$\text{MSBE}(\mathbf{w}) = \|\overline{\delta}_{\mathbf{w}}\|_\mu^2 \tag{11.19}$$

定义一个 Bellman 操作:

$$\begin{aligned} &(B_\pi v)(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')], \\ &\forall s \in \mathcal{S}, \quad \forall v : \mathcal{S} \to \mathbb{R} \end{aligned} \tag{11.20}$$

则 Bellman 误差向量可写作:

$$\overline{\delta}_{\mathbf{w}} = B_\pi v_{\mathbf{w}} - v_{\mathbf{w}} \tag{11.21}$$

其 Bellman 均方投影误差 (Mean Square Projected Bellman Error, MSPBE):

$$\text{MSPBE}(\mathbf{w}) = \|\Pi\overline{\delta}_{\mathbf{w}}\|_\mu^2 \tag{11.22}$$

11.5 Stochastic Gradient Descent in the Bellman Error

随机梯度下降 (Stochastic Gradient Descent, SGD) 具有较好的鲁棒性，能够较快地收敛，但是速度慢于半随机梯度下降 (semi-SGD)。

在 TD 学习中，定义均方 TD 误差 (Mean Squared TD Error, MSTDE):

$$\begin{aligned}\text{MSTDE}(\mathbf{w}) &= \sum_{s\in\mathcal{S}} \mu(s)\mathbb{E}[\delta_t^2|S_t=s, A_t\sim\pi] \\ &= \sum_{s\in\mathcal{S}} \mu(s)\mathbb{E}[\rho_t\delta_t^2|S_t=s, A_t\sim b] \\ &= \mathbb{E}_b[\rho_t\delta_t^2]\end{aligned} \tag{11.23}$$

那么对应的更新步骤可以写为:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla(\rho_t\delta_t^2) \\ &= \mathbf{w}_t - \alpha\rho_t\delta_t\nabla\delta_t \\ &= \mathbf{w}_t + \alpha\rho_t\delta_t\left(\nabla\hat{v}(S_t,\mathbf{w}_t) - \gamma\nabla\hat{v}(S_t,\mathbf{w}_t)\right)\end{aligned} \tag{11.24}$$

这个算法被称为 naive residual-gradient 算法。虽然这个算法有较强的鲁棒性，但是常常收敛不到较好的值，对此比较好的是采用 Bellman 误差，对应的更新公式可以写为:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla(\mathbb{E}_\pi[\delta_t]^2) \\ &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla(\mathbb{E}_b[\rho_t\delta_t]^2) \\ &= \mathbf{w}_t - \alpha\mathbb{E}_b[\rho_t\delta_t]\nabla\mathcal{E}_b[\rho_t\delta_t] \\ &= \mathbf{w}_t - \alpha\mathbb{E}_b[\rho_t(R_{t+1}+\gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w}))]\mathbb{E}_b[\rho_t\nabla\delta_t] \\ &= \mathbf{w}_t + \alpha\left[\mathbb{E}_b[\rho_t(R_{t+1}+\gamma\hat{v}(S_{t+1},\mathbf{w})) - \hat{v}(S_t,\mathbf{w})\right]\left[\nabla\hat{v}(S_t,\mathbf{w}) - \gamma\mathbb{E}_b[\rho_t\nabla\hat{v}(S_{t+1},\mathbf{w})]\right]\end{aligned} \tag{11.25}$$

这种算法也称为残差梯度 (residual gradient)

11.6 Learnability of the Bellman Error

如果不能从现有的特征向量、动作或者奖励中进行计算和估计，那么我们称之为不能学习的 (not *learnable*). Bellman 误差就是一个不可学习过程。

首先定义两个马尔可夫奖励过程 (Markov reward processes, MRP):

其中数字代表的就是 reward，所有状态都是等概率的。看右边的图，转移回本状态 reward=0，转移到另一状态 reward=2，那么平均 reward=1，均方误差在两个图上为 0. 定义均方回报误差 (Mean Square Return Error, MSRE):

$$\begin{aligned}\text{MSRE}(\mathbf{w}) &= \mathbb{E}\left[(G_t - \hat{v}(S_t, \mathbf{w}))^2\right] \\ &= \text{MSVE}(\mathbf{w}) + \mathbb{E}\left[(G_t - v_\pi(S_t))^2\right]\end{aligned} \tag{11.26}$$

MSVE 和 MSRE 两者都不倚重权重参数向量，这说明了均方价值误差 MSVE 是不可学习的，而 MSRE 是可学习的。

总之均方 Bellman 误差 MSBE 是不可学习的，不能被可观察到的数值或者是参数中进行估计和计算，说明通过最小化 MSBE 来收敛到较优解是不可行的。我们应该研究的是均方 Bellman 投影误差 (MSPBE).

11.7 Gradient-TD Methods

我们通过随机梯度下降 (SGD) 来最小化均方 Bellman 投影误差 (MSPBE), 梯度 TD 算法在离策略和非线性函数逼近中具有较好的鲁棒性，在 TD 算法中引入随机梯度下降之后，算法复杂度将由 $O(d^2)$ 下降为 $O(d)$。公式可

以写为：

$$\begin{aligned}\text{MSPBE}(\mathbf{w}) &= \|\Pi\overline{\delta}_{\mathbf{w}}\|_{\mu}^{2} \\ &= (\Pi\overline{\delta}_{\mathbf{w}})^{T} D\Pi\overline{\delta}_{\mathbf{w}} \\ &= \overline{\delta}_{\mathbf{w}}^{T}\Pi^{T} D\Pi\overline{\delta}_{\mathbf{w}} \\ &= \overline{\delta}_{\mathbf{w}}^{T} D\mathbf{X}(\mathbf{X}^{T}D\mathbf{X})^{-1}\mathbf{X}^{T}D\overline{\delta}_{\mathbf{w}} \\ &= (\mathbf{X}^{T}D\overline{\delta}_{\mathbf{w}})^{T}(\mathbf{X}^{T}D\mathbf{X})^{-1}(\mathbf{X}^{T}D\overline{\delta}_{\mathbf{w}})\end{aligned} \tag{11.27}$$

其中：

$$\Pi^{T}D\Pi = D\mathbf{X}(\mathbf{X}^{T}DX)^{-1}\mathbf{X}^{T}D \tag{11.28}$$

那么 $\mathbf{w}$ 的梯度可以写为：

$$\nabla\text{MSPBE}(\mathbf{w}) = 2\nabla\left[\mathbf{X}^{T}D\overline{\delta}_{\mathbf{w}}\right]^{T}(\mathbf{X}^{T}D\mathbf{X})^{-1}(\mathbf{X}^{T}D\overline{\delta}_{\mathbf{w}}) \tag{11.29}$$

假设 μ 是访问行为策略的分布，那么：

$$\mathbf{X}^{T}D\overline{\delta}_{\mathbf{w}} = \sum_{s}\mu(s)\mathbf{x}(s)\overline{\delta}_{\mathbf{w}}(s) = \mathbb{E}[\mathbf{x}_{t}\rho_{t}\delta_{t}] \tag{11.30}$$

对应的梯度：

$$\begin{aligned}\nabla\mathbb{E}[\mathbf{x}_{t}\rho_{t}\delta_{t}]^{T} &= \mathbb{E}\left[\rho_{t}\nabla\delta_{t}^{T}\mathbf{x}_{t}^{T}\right] \\ &= \mathbb{E}\left[\rho_{t}\nabla(R_{t+1}+\gamma\mathbf{w}^{T}\mathbf{x}_{t+1}-\mathbf{w}^{T}\mathbf{x}_{t})^{T}\mathbf{x}_{t}^{T}\right] \\ &= \mathbb{E}\left[\rho_{t}(\gamma\mathbf{x}_{t+1}-\mathbf{x}_{t})\mathbf{x}_{t}^{T}\right]\end{aligned} \tag{11.31}$$

综上：

$$\nabla\text{MSPBE}(\mathbf{w}) = 2\mathbb{E}\left[\rho_{t}(\gamma\mathbf{x}_{t+1}-\mathbf{x}_{t})\mathbf{x}_{t}^{T}\right]\mathbb{E}\left[\mathbf{x}_{t}\mathbf{x}_{t}^{T}\right]^{-1}\mathbb{E}[\mathbf{x}_{t}\rho_{t}\delta_{t}] \tag{11.32}$$

上面的公式说明，即使是 MSPBE，也是需要下一个特征向量 $\mathbf{w}_{t+1}$。导致我们不能够简单地进行采样，求取期望。

在梯度 TD 算法中，分别存储一些估计，并且与采样的样本进行结合，是一种解决问题的思路。我们将要存储的信息设为 $\mathbf{v}$:

$$\mathbf{v} \approx \mathbb{E}[\mathbf{x}_{t}\mathbf{x}_{t}^{T}]^{-1}\mathbb{E}[\mathbf{x}_{t}\rho_{t}\delta_{t}] \tag{11.33}$$

最小化方差误差 (Least Mean Square,LMS) 为:

$$(\mathbf{v}^T\mathbf{x}_t - \rho_t\delta_t)^2 \tag{11.34}$$

那么特征向量的更新公式可以写为:

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \beta\rho_t(\delta_t - \mathbf{v}_t^T\mathbf{x}_t)\mathbf{x}_t \tag{11.35}$$

那么权重向量的更新就可以写为:

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla\text{MSPBE}(\mathbf{w}_t) \\
&= \mathbf{w}_t - \frac{1}{2}\alpha 2\mathbb{E}\left[\rho_t(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t)\mathbf{x}_t^T\right]\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}[\mathbf{x}_t\rho_t\delta_t] \\
&= \mathbf{w}_t + \alpha\mathbb{E}\left[\rho_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}_t^T\right]\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}[\mathbf{x}_t\rho_t\delta_t] \\
&= \mathbf{w}_t + \alpha\mathbb{E}\left[\rho_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}_t^T\right]\mathbf{v}_t \\
&= \mathbf{w}_t + \alpha\rho_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}_t^T\mathbf{v}_t
\end{aligned} \tag{11.36}$$

这个算法被称为 $GTD2$, 对于该算法，还可以有一点提升:

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\mathbb{E}\left[\rho_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}_t^T\right]\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}[\mathbf{x}_t\rho_t\delta_t] \\
&= \mathbf{w}_t + \alpha\left(\mathbb{E}\left[\rho_t\mathbf{x}_t\mathbf{x}_t^T\right] - \gamma\mathbb{E}\left[\rho_t\mathbf{x}_{t+1}\mathbf{x}_t^T\right]\right)\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}\left[\mathbf{x}_t\rho_t\delta_t\right] \\
&= \mathbf{w}_t + \alpha\left(\mathbb{E}\left[\mathbf{x}_t\rho_t\delta_t\right] - \gamma\mathbb{E}\left[\rho_t\mathbf{x}_{t+1}\mathbf{x}_t^T\right]\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}\left[\mathbf{x}_t\rho_t\delta_t\right]\right) \\
&= \mathbf{w}_t + \alpha\left(\mathbb{E}\left[\mathbf{x}_t\rho_t\delta_t\right] - \gamma\mathbb{E}\left[\rho_t\mathbf{x}_{t+1}\mathbf{x}_t^T\right]\mathbf{v}_t\right) \\
&= \mathbf{w}_t + \alpha\rho_t\left(\delta_t\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\mathbf{x}_t^T\mathbf{v}_t\right)
\end{aligned} \tag{11.37}$$

这个算法被称为 $GTD(0)$ 或者 TDC。

GTD2 和 TDC 都包括两个学习进程，主要第一个的 $\mathbf{w}$ 和第二个 $\mathbf{v}$。如果主要的第一个学习有第二个影响，而第二个不会比第一个先进行，那么这种关系就称为级联 (cascade)。

综上，梯度 TD 算法 (GTD) 是当前最好理解并且最稳定的使用最广泛的离策略。

11.8 Emphatic-TD Methods

重要性 TD 算法 (Emphatic-TD Methods,ETD)，其中 one-step emphatic-TD 算法的参数可以写为：

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha M_t \rho_t \delta_t \nabla \hat{v}(S_t, \mathbf{w}_t) \\ M_t &= \gamma \rho_{t-1} M_{t-1} + I_t\end{aligned} \tag{11.38}$$

I_t 表示 t 时刻的兴趣 (interest)，是一个随机的值；M_t 表示重要性 (emphasis)，其中初始化 $M_{t-1} = 0$。

在接下来的实验中，ETD 被证明出分歧 (variance) 太大，而无法进行实用。

11.9 Reducing Variance

off-policy 中分歧是不可避免的问题，在控制分歧，特别是离策略中需要十分谨慎。众所周知，重要性采样中重要性比率 ratios 通常期望为 1，但是在实际中 $ratios \gg 1$ 或者 $ratios \to 0$。理想中的 ratios 是互相之间无关联的。但是实际上就是这些相关联导致了目标策略和行为策略之间的分歧。

随机梯度下降 (SGD) 通过多次小步的计算得到良好的梯度，这样可以有效的减少分歧，但是 SGD 必须拥有比较大的样本才容易收敛，若样本过于单一的话，得到的结果也将变得不完善了。但是要注意的是，如果步长太小了的话，会导致预期的步骤变得非常小，计算的效率会特别地低。

Chapter 12

Eligibility Traces

12.1 The λ-return

首先给出 n-step 回报公式:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}), \quad (12.1)$$

在这里,平均回报可以由一半两步回报和一半四步回报构成,即 $G = \frac{1}{2}G_{t:t+2} + \frac{1}{2}G_{t:t+4}$。这种更新方式称为复合更新 (compound update), 由此引出的算法称为 TD(λ) 算法, 这个平均包含 n 步回报, 权重比例为 $\lambda^{n-1}, \lambda \in [0,1]$, 加上系数 $(1-\lambda)$ 保证和为 1(极限求和)。其公式可以写作:

$$G_t^\lambda \doteq (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \quad (12.2)$$

对其进行一定的分步计算:

$$G_t^\lambda \doteq (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t \quad (12.3)$$

由此我们可以引出离线 λ 回报算法 (off-line λ-return algorithm), 在该算法进行中, 它不会对权重向量进行改变, 根据半梯度 (semi-gradient) 思想, 有:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[G_t^{\lambda s} - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t), \quad (12.4)$$

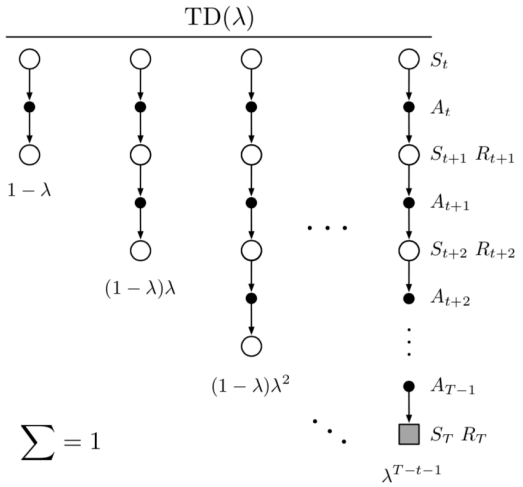


Figure 12.1: The backup digram for TD(λ). If $\lambda = 0$, then the overall backup reduces to its first component, the one-step TD backup, whereas if $\lambda = 1$, then the overall backup reduces to its last component, the Monte Carlo backup.

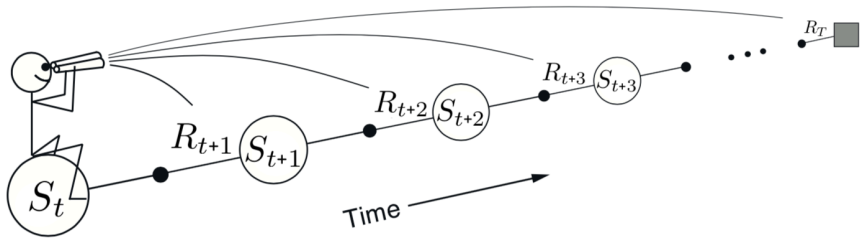


Figure 12.4: The forward view. We decide how to update each state by looking forward to future rewards and states.

12.2 TD(λ)

TD(λ) 算法从三个方面提升离线 λ 回报算法：

1. 在一次模拟的每一步，TD(λ) 都会更新权重向量，而不是模拟结束后再进行更新，这样可以提高当前的估计速度。
2. 这样计算的开销将会等量分布到估计每一时刻，而不是集中在估计结束时候。
3. 除了估计问题，TD(λ) 还可以应用于持续性问题 (continuing problems)。

在函数值逼近中，资格轨迹 (eligibility trace) 向量写为 $\mathbf{e}_t \in \mathbb{R}^d$, 它和权重向量组成形式是一样的。初始值一般设为 0，以值梯度 (value gradient) 进行增量更新，以 $\gamma\lambda$ 作为新的折扣：

$$\begin{aligned} &\mathbf{e}_{-1} \doteq \mathbf{0}, \\ &\mathbf{e}_t \doteq \gamma\lambda\mathbf{e}_{t-1} + \nabla\hat{v}(S_t, \mathbf{w}_t), \quad 0 \le t \le T. \end{aligned} \tag{12.5}$$

对应的 TD error 可以写为：

$$\delta_t \doteq R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_{t+1}, \mathbf{w}_t) \tag{12.6}$$

权重向量的更新公式可以写为：

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t\mathbf{e}_t \tag{12.7}$$

那么半梯度 TD(λ) 算法可以概括为：

Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

1. 输入：需要被估计的策略 π;
2. 输入：一个误差函数 $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ 且 $\hat{v}(\text{terminal}, \cdot) = 0$;
3. 随机地初始化价值函数权重 $\mathbf{w}$，如 $\mathbf{w} = \mathbf{0}$;
4. 对于每条轨迹，重复如下步骤直到 S' 到达中断状态：

(a) 初始化 S;

(b) n 维向量 $\mathbf{e} \leftarrow \mathbf{0}$;

(c) 对于一条轨迹的每一步，重复如下步骤:

i. 选择 $A \sim \pi(\cdot|S)$;

ii. 执行动作 A，观察 R, S';

iii. $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e} + \nabla\hat{v}(S, \mathbf{w})$;

iv. $\delta \leftarrow R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$;

v. $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{e}$;

vi. $S \leftarrow S'$;

对应的示意图如下:

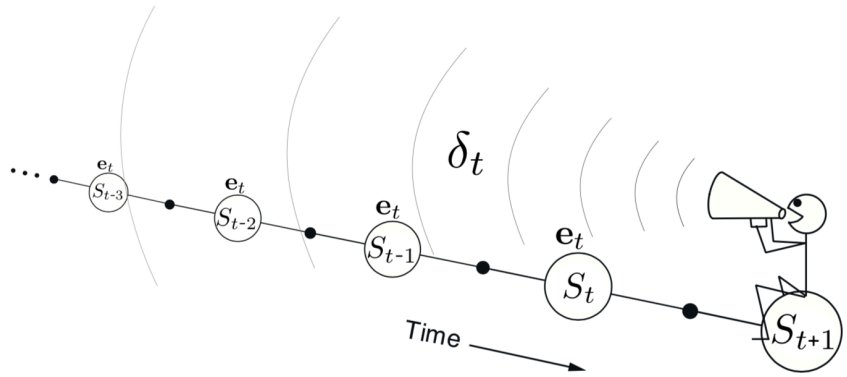


Figure 12.5: The backward or mechanistic view. Each update depends on the current TD error combined with eligibility traces of past events.

12.3 n-step Truncated λ-return Methods

在持续性环境下，λ 回报只有在模拟结束后才能获取，所以我们要定义一个缩短 (truncated) 了的 λ 回报, 即:

$$G_{t:h}^{\lambda} \doteq (1-\lambda) \sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{h-t-1} G_{t:h} \tag{12.8}$$

是一个与 12.3很相近的公式，定义短期 TD(λ) 为 TTD(λ)：

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha \left[G^{\lambda}_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1}) \right] \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}), \tag{12.9}$$

这个算法可以减少每一步计算的开销，但是无法减少计算需要的空间，那么 k 步 λ 回报可以被写成：

$$\begin{aligned} G^{\lambda}_{t:t+k} &= \hat{v}(S_t, \mathbf{w}_{t-1}) + \sum_{i=t}^{t+k-1} (\gamma\lambda)^{i-t} \delta'_t, \\ \delta'_t &\doteq R_{i+1} + \gamma \hat{v}(S_{i+1}, \mathbf{w}_i) - \hat{v}(S_i, \mathbf{w}_{i-1}). \end{aligned} \tag{12.10}$$

12.4 Redoing Updates: The Online λ-return Algorithm

我们希望缩减参数 (truncation parameter)n 尽可能地大，以便于结果接近离线 λ 回报算法；但在同时，我们也希望 n 尽可能小，这样可以使得计算变得比较快。

解决这个矛盾的方法：在每一次模拟时，一旦获得新的数据增量，立马回到模拟初始处并重新进行所有的更新。在这个思想下，更新权重的公式可以写为：

$$\mathbf{w}^h_{t+1} \doteq \mathbf{w}^h_t + \alpha \left[G^{\lambda}_{t:h} - \hat{v}(S_t, \mathbf{w}^h_t) \right] \nabla \hat{v}(S_t, \mathbf{w}^h_t) \tag{12.11}$$