

强化学习

第六讲：价值函数逼近

教师：赵冬斌 朱圆恒

中国科学院大学
中国科学院自动化研究所



回顾

求解贝尔曼最优方程条件：

- 求解非线性算子 \max

- 模型已知

- 足够的计算空间

回顾

求解贝尔曼最优方程条件：

- 求解非线性算子 \max
 - 动态规划：价值迭代，策略迭代
 - 模型已知
- 足够的计算空间

回顾

求解贝尔曼最优方程条件：

- 求解非线性算子 \max
 - 动态规划：价值迭代，策略迭代
- 模型已知
 - 预测 (prediction): 蒙特卡洛方法, 时间差分学习, $TD(\lambda)$
 - 控制 (control): Sarsa, Sarsa(λ), Q-学习
- 足够的计算空间

回顾

求解贝尔曼最优方程条件：

- 求解非线性算子 \max
 - 动态规划: 价值迭代, 策略迭代
- 模型已知
 - 预测 (prediction): 蒙特卡洛方法, 时间差分学习, $TD(\lambda)$
 - 控制 (control): Sarsa, Sarsa(λ), Q-学习
- 足够的计算空间
 - 本节内容: **函数逼近**
 - 和之前的方法结合起来

- 有限 MDPs 使用 **查表法** 存储每个状态或状态 - 动作的价值和策略
 - $V(s)$, $Q(s, a)$, $\pi(a|s)$, $\forall s, a$
- 强化学习会面对 **大规模 (large-scale)** 的问题, 例如
 - 西洋双陆棋戏: 10^{20} 状态
 - 围棋: 10^{170} 状态
 - 直升机: 连续状态空间

- 有限 MDPs 使用 **查表法** 存储每个状态或状态 - 动作的价值和策略
 - $V(s)$, $Q(s, a)$, $\pi(a|s)$, $\forall s, a$
- 强化学习会面对 **大规模 (large-scale)** 的问题, 例如
 - 西洋双陆棋戏: 10^{20} 状态
 - 围棋: 10^{170} 状态
 - 直升机: 连续状态空间
- 如何将之前的 **动态规划**, **预测学习**, **控制学习** 方法扩展到大规模问题上?

函数逼近器

- 对于一个复杂的函数 $f(\mathbf{x})$
- 使用另外一种结构化的函数 $g(\mathbf{x}|\mathbf{w})$ 近似, \mathbf{w} 代表逼近函数的关键参数
- 相比于原函数, 逼近函数结构更简单, 需要调整的参数量更少
- 我们统一用 Π 代表从 $f(\mathbf{x})$ 函数空间到 $g(\mathbf{x}|\mathbf{w})$ 函数空间的 **最佳匹配 (best fit)**

$$g(\mathbf{x}|\mathbf{w}^*) = \Pi(f(\mathbf{x}))$$

也就是说 $g(\mathbf{x}|\mathbf{w}^*)$ 最能近似 $f(\mathbf{x})$

举例：泰勒展式

- 对任意的 $f(x)$, 在 x_0 点使用泰勒展式逼近

$$g_0(x) = f(x_0)$$

$$g_1(x) = f(x_0) + f'(x_0)(x - x_0)$$

$$g_2(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$$

⋮

$$g_n(x) = f(x_0) + f'(x_0)(x - x_0) + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

- 逼近函数 $g(x)$ 选择是多项式的表示形式

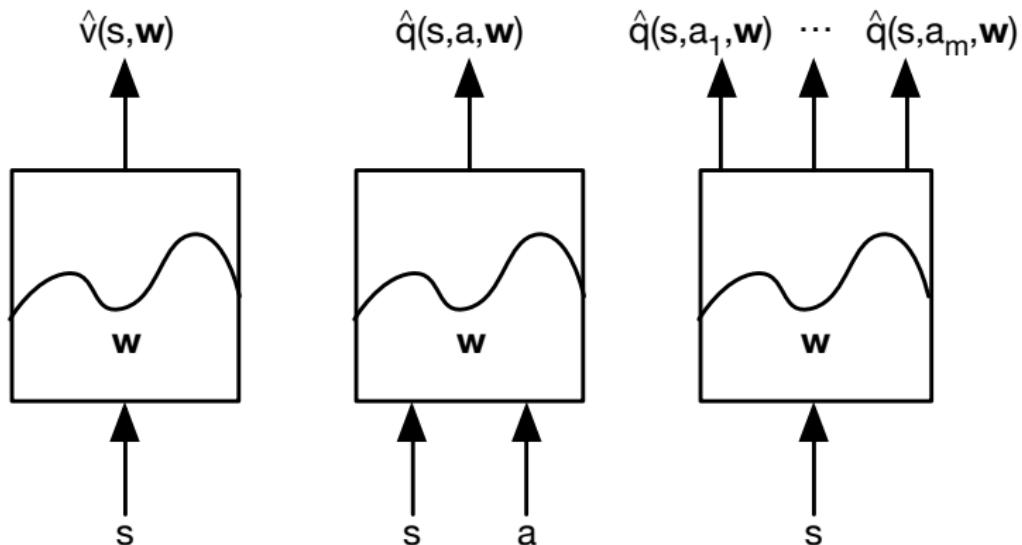
$$g(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)^2 + \dots$$

- 可调整的参数是多项式的系数 c_0, c_1, c_2, \dots

- 泰勒展式决定了最佳逼近原函数的系数是 $c_n = \frac{f^{(n)}(x_0)}{n!}$

- 大规模强化学习使用逼近器的好处：
 - **计算复杂度**：使用更少的参数逼近价值函数（而不是对每个状态，动作存储价值）
 - **泛化能力**：更少样本就能学到较精确的价值函数（而不是遍历整个状态，动作空间）
 - **多样性**：多种特征表示和逼近器结构（选择针对问题属性的逼近器，e.g. AlphaGo 使用 CNN）

价值函数逼近的类型



- 1 V 函数逼近: 输入 s , 输出 $V(s)$
- 2 Q 函数逼近: 输入 (s, a) , 输出 $Q(s, a)$
- 3 Q 函数逼近: 输入 s , 输出所有的 $Q(s, a), \forall a$

使用哪种函数逼近器？

- 存在许多种类型的函数逼近器，例如：
 - 特征函数的线性组合
 - 神经网络
 - 决策树
 - 最近邻
 - 傅立叶/小波基函数
 - ...

使用哪种函数逼近器？

- 存在许多种类型的函数逼近器，例如：
 - 特征函数的线性组合
 - 神经网络
 - 决策树
 - 最近邻
 - 傅立叶/小波基函数
 - ...
- 我们通常考虑的是 可微 (differentiable) 的函数逼近器

线性函数逼近

线性价值函数逼近 Linear VFA

- 用 特征向量 (Feature Vector) 代表某一状态

$$\mathbf{x}(s) = \begin{pmatrix} \mathbf{x}_1(s) \\ \vdots \\ \mathbf{x}_n(s) \end{pmatrix}$$

- 用 一组特征的线性组合 逼近价值函数

$$\hat{V}(s, \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w} = \sum_{j=1}^n \mathbf{x}_j(s) w_j$$

w_j 代表特征的 权重 , \mathbf{w} 是 权重向量

- 特征反映了状态是否具有某种模式
 - 机器人运动控制中一个特定的姿态是特征
 - 股票市场中的成交量、收盘价是特征
 - 围棋中各种棋子摆放组合是特征

均方误差 Mean Squared Error, MSE

- 均方误差 (Mean Squared Error, MSE) 描述逼近函数和真实价值函数之间的误差

$$\begin{aligned} J(\mathbf{w}) &= \mathbb{E}_{\pi} \left[(V_{\pi}(s) - \hat{V}(s, \mathbf{w}))^2 \right] \\ &= \int_s d(s) (V_{\pi}(s) - \hat{V}(s, \mathbf{w}))^2 \end{aligned}$$

其中 d 代表了智能体在策略 π 下的状态分布

- $J(\mathbf{w})$ 也称为 期望损失 (Expected Loss)
- 线性逼近器的最佳匹配满足

$$\begin{aligned} \hat{V}(\mathbf{w}^*) &= \Pi(V_{\pi}) \\ s.t. \quad \mathbf{w}^* &= \arg \min_{\mathbf{w}} \mathbb{E}_{\pi} \left[(V_{\pi}(s) - \mathbf{x}(s)^T \mathbf{w})^2 \right] \end{aligned}$$

- 对线性逼近器，用样本近似期望损失

$$\begin{aligned} J(\mathbf{w}) &= \mathbb{E}_\pi \left[(V_\pi(s) - \hat{V}(s, \mathbf{w}))^2 \right] \\ &\approx \frac{1}{M} \sum_{m=1}^M (V_\pi(s_m) - \mathbf{x}(s_m)^T \mathbf{w})^2 \end{aligned}$$

- 二次损失函数 的最小点在一阶偏导等于 0 的点

$$\min_{\mathbf{w}} J(\mathbf{w}) \Rightarrow \nabla_{\mathbf{w}} J(\mathbf{w}) = 0$$

$$\begin{aligned} \Rightarrow 0 &= \sum_{m=1}^T \mathbf{x}(s_m) (V_\pi(s_m) - \mathbf{x}(s_m)^T \mathbf{w}^*) \\ \Rightarrow \mathbf{w}^* &= \left(\sum_{m=1}^T \mathbf{x}(s_m) \mathbf{x}(s_m)^T \right)^{-1} \sum_{m=1}^T \mathbf{x}(s_m) V_\pi(s_m) \end{aligned}$$

- 最小二乘法优点：计算过程简单，一次性求解最佳权重
- 但是需要对 $n \times n$ 的矩阵求逆，计算复杂度是 $\mathcal{O}(n^3)$
- 当特征空间比较大时，计算量大，矩阵可能不满秩，容易有误差

- $J(\mathbf{w})$ 是任意 **可微** 逼近器 (线性/非线性) 的损失函数
- 定义 $J(\mathbf{w})$ 的梯度

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

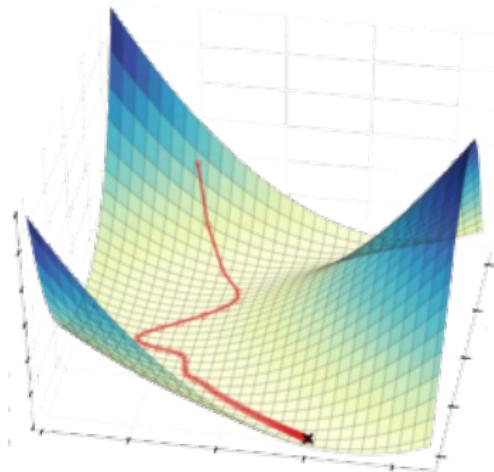
- 当权重 \mathbf{w} 向 **梯度反方向** 调整时, 能够降低 $J(\mathbf{w})$

$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

α 是调整步长 (step-size)

■ 梯度下降训练过程:

- 1 初始化一组权重 \mathbf{w}
- 2 计算当前权重下的损失 $J(\mathbf{w})$ 和梯度 $\nabla_{\mathbf{w}} J(\mathbf{w})$
- 3 更新 $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
- 4 go back to step 2



- 完全的权重更新公式计算量大

$$\begin{aligned}\Delta \mathbf{w} &= -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}) = \alpha \mathbb{E}[(V_{\pi}(s) - \hat{V}(s, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s, \mathbf{w})] \\ &\approx \frac{\alpha}{M} \sum_{m=1}^M (V_{\pi}(s_m) - \hat{V}(s_m, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_m, \mathbf{w})\end{aligned}$$

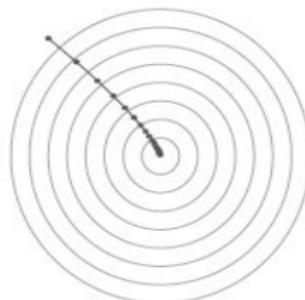
- 随机梯度下降法：每次用单个样本计算梯度

$$\Delta \mathbf{w} = \alpha (V_{\pi}(s_m) - \hat{V}(s_m, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_m, \mathbf{w})$$

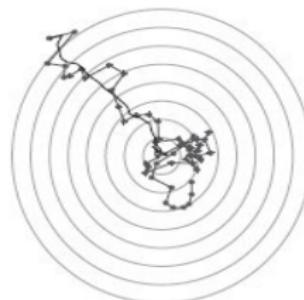
- 目标函数已知前提下 SGD 的期望是无偏的

■ SGD 更新过程

- 1 准备数据集 $\mathcal{D} = \{\langle s_1, V_\pi(s_1) \rangle, \dots, \langle s_M, V_\pi(s_M) \rangle\}$
- 2 初始化权重 \mathbf{w}
- 3 随机选择一组数据 $\langle s_m, V_\pi(s_m) \rangle$ 计算
$$\Delta \mathbf{w} = \alpha (V_\pi(s_m) - \hat{V}(s_m, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_m, \mathbf{w})$$
- 4 更新 $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
- 5 返回步骤 3



full gradient descent



stochastic gradient descent

常见的特征表示方法

查表法

- 查表法可以看作是线性价值函数逼近器的一种特例
- 使用 **查表特征**

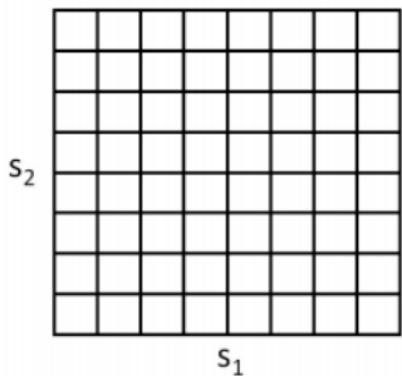
$$\mathbf{x}^{table}(s) = \begin{pmatrix} \mathcal{I}(s = s_1) \\ \vdots \\ \mathcal{I}(s = s_n) \end{pmatrix}$$

- 每个状态对应一个权重 \mathbf{w}_j 代表该状态的价值

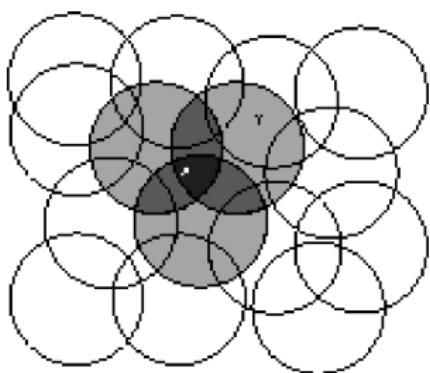
$$\hat{V}(s, \mathbf{w}) = \begin{pmatrix} \mathcal{I}(s = s_1) \\ \vdots \\ \mathcal{I}(s = s_n) \end{pmatrix}^T \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

离散化法

- 对连续的状态或动作空间，离散化法是最简单的一种特征表示
- 将连续的空间 \mathcal{S} 划分成 非重叠的，相邻的子空间 $\{\mathcal{S}_i\}$
- 状态在同一个子空间上具有相同的价值



$$\hat{V}(s, \mathbf{w}) = \begin{pmatrix} \mathcal{I}(s \in \mathcal{S}_1) \\ \vdots \\ \mathcal{I}(s \in \mathcal{S}_n) \end{pmatrix}^T \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

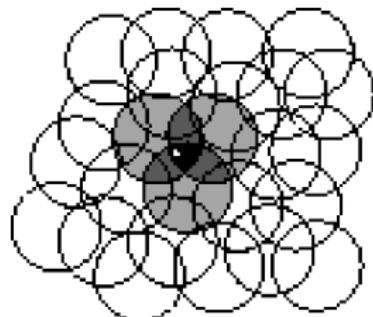


- 同样是考虑连续状态空间
- 一个特征对应空间中的一个圆形区域
 - 如果状态在圆内, 特征为 1
 - 如果状态在圆外, 特征为 0
- 随机选取一组圆圈的圆心位置
- 所有的圆圈组成了一组特征向量

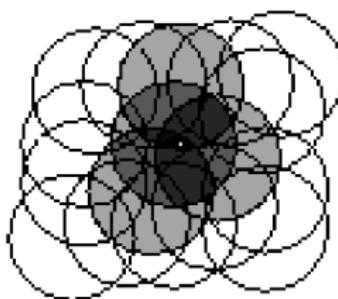
$$\mathbf{x}(s) = \begin{pmatrix} \mathcal{I}(s \text{ is in circle 1}) \\ \vdots \\ \mathcal{I}(s \text{ is in circle n}) \end{pmatrix}$$

- 与离散化方法不同, 特征是可以重叠的

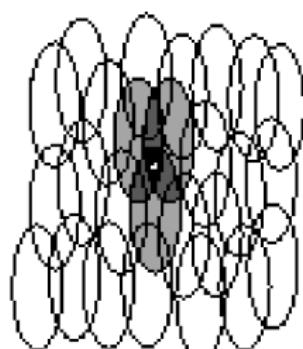
- 改变圆圈的数量
- 改变圆圈的范围
- 甚至改变圆圈的形状, 都能获得不同的特征表示



a) Narrow Generalization



b) Broad Generalization

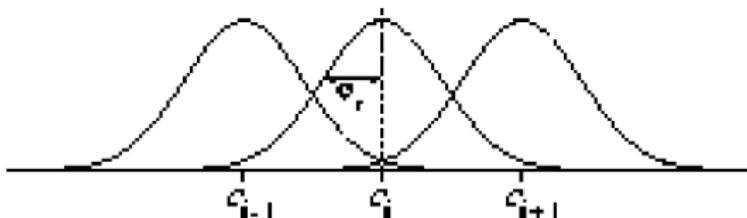


c) Asymmetric Generalization

- 离散化和粗糙编码都是 **二值化** 的特征表示
- 径向基函数是粗糙编码向 **连续型特征表示** 的泛化
- 特征值可以是 $[0,1]$ 之间的任意值, 代表属于该特征的程度
- 最典型的 RBF 是 **高斯函数**
 - 一维状态 s 的高斯函数:

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

- c_i 和 σ_i 分别对应高斯的中心点和宽度



■ 多维状态的高斯函数

$$\phi_i(s) = \exp\left(-\frac{1}{2}[s - c_i]^T B_i^{-1}[s - c_i]\right)$$

- c_i 是中心向量, B_i 是协方差矩阵
- 定义一组 RBF $\{(c_i, B_i)\}$, 组成特征向量

$$\mathbf{x}(s) = \begin{pmatrix} \phi_1(s) \\ \vdots \\ \phi_n(s) \end{pmatrix}$$

- 有时会对特征向量 归一化

$$\mathbf{x}(s) = \begin{pmatrix} \phi_1(s) / \sum_i \phi_i(s) \\ \vdots \\ \phi_n(s) / \sum_i \phi_i(s) \end{pmatrix}$$

这样 $\sum_i \mathbf{x}_i(s) = 1$

价值迭代 + 离散化方法

回顾：价值迭代

- 基于 Q 函数的价值迭代：

$$Q_{i+1}(s, a) = [\mathcal{T}(Q_i)](s, a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_i(s', a')$$

$$\pi^*(s) = \arg \max_a Q_\infty(s, a)$$

- 无法直接求解 连续空间 MDPs 问题：

- 无穷个 (s, a) , 无法用查表法定义 $Q(s, a)$
- 无法在连续动作空间上求 max

回顾：价值迭代

- 基于 Q 函数的价值迭代：

$$Q_{i+1}(s, a) = [\mathcal{T}(Q_i)](s, a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_i(s', a')$$

$$\pi^*(s) = \arg \max_a Q_\infty(s, a)$$

- 无法直接求解 连续空间 MDPs 问题：

- 无穷个 (s, a) , 无法用查表法定义 $Q(s, a)$
- 无法在连续动作空间上求 max

- 解决办法：

- 将连续的状态空间离散化成 有限状态子集 $\{S_1, \dots, S_K\}$
- 在连续的动作空间上选择有限个代表性的动作，构成 有限动作集 $\{\mathcal{A}_1, \dots, \mathcal{A}_L\}$

- 对每个状态子空间和离散动作定义一个 $Q_{k,l}$ 值
- 子空间的所有状态在同一个离散动作下共享同一个 Q 值

$$Q_{k,l} = Q(s, a), \quad s \in \mathcal{S}_k, a = \mathcal{A}_l$$

- 将连续空间 MDPs 转化成有限 MDPs
- 价值迭代时也只需要更新 $\{Q_{k,l}\}$ 的值

- 我们在每个状态子集上选择一个代表性的状态 $\bar{s}_k \in \mathcal{S}_k$
 - 通常选状态子集的中心点
- 和离散化动作 \mathcal{A}_l 一起代入价值迭代算子

$$[\mathcal{T}(Q)](s, a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_i(s', a'), \quad s = \bar{s}_k, a = \mathcal{A}_l$$

- 我们在每个状态子集上选择一个代表性的状态 $\bar{s}_k \in \mathcal{S}_k$
 - 通常选状态子集的中心点
- 和离散化动作 \mathcal{A}_l 一起代入价值迭代算子

$$[\mathcal{T}(Q)](s, a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_i(s', a'), \quad s = \bar{s}_k, a = \mathcal{A}_l$$

- 对连续空间 MDPs, 奖励函数和转移函数通常是确定型的
- $\mathcal{R}_{k,l} = \mathcal{R}(\bar{s}_k, \mathcal{A}_l)$, $s' = \mathcal{P}(\bar{s}_k, \mathcal{A}_l)$
- 并且我们可以确定 s' 所在的状态子集, $s' \in \mathcal{S}_{k'}$
- 并将 a' 最大化范围限定在离散动作集上 $s' \in \{\mathcal{A}_1, \dots, \mathcal{A}_L\}$
- 离散化的价值迭代变成

$$Q_{k,l}^{(i+1)} = \mathcal{R}_{k,l} + \gamma \max_{l'} Q_{k',l'}^{(i)}$$

离散化的价值迭代算法

- 1: 将状态空间离散化成状态子集 $\{\mathcal{S}_k\}, k = 1, \dots, K$
- 2: 定义有限动作集 $\{\mathcal{A}_l\}, l = 1, \dots, L$
- 3: 初始化 $Q_{k,l}^{(0)}$ (e.g. $Q_{k,l}^{(0)} = 0$), $i = 0$
- 4: **repeat** {在第 i 次迭代}
- 5: 更新 Q 值

$$Q_{k,l}^{(i+1)} = \mathcal{R}_{k,l} + \gamma \max_{l'} Q_{k',l'}^{(i)}, \quad \forall k, l$$

- 6: $i \leftarrow i + 1$
- 7: **until** $\|Q^{(i)} - Q^{(i-1)}\| \leq \varepsilon$

举例：直流电机

- 二维状态，一维动作的连续状态转移函数

$$s_{t+1} = \mathcal{P}(s_t, a_t) = \begin{bmatrix} 1 & 0.0049 \\ 0 & 0.9540 \end{bmatrix} s_t + \begin{bmatrix} 0.0021 \\ 0.8505 \end{bmatrix} a_t$$

- $s_t(1) = \alpha \in [-\pi, \pi]$ rad 电机角度
- $s_t(2) = \dot{\alpha} \in [-16\pi, 16\pi]$ rad/s 角速度
- $a_t \in [-10, 10]$ V 输入电压

- 奖励函数

$$r_{t+1} = \mathcal{R}(s_t, a_t) = -c_1 s_t(1)^2 - c_2 s_t(2)^2 - c_3 a_t^2,$$

$$c_1 = 5, c_2 = 0.01, c_3 = 0.01$$

- 折扣因子 $\gamma = 0.95$

■ 状态空间离散化

- 将角度 α 在 $[-\pi, \pi]$ 范围均匀划分成 20 份

$$\left\{ \left[-\pi + \frac{2\pi}{20}i, -\pi + \frac{2\pi}{20}(i+1) \right] \right\}, i = 0, \dots, 19$$

- 将角速度 $\dot{\alpha}$ 在 $[-16\pi, 16\pi]$ 范围均匀划分成 20 份

$$\left\{ \left[-16\pi + \frac{32\pi}{20}j, -16\pi + \frac{32\pi}{20}(j+1) \right] \right\}, j = 0, \dots, 19$$

- 状态空间离散化成 $20 * 20 = 400$ 个子集

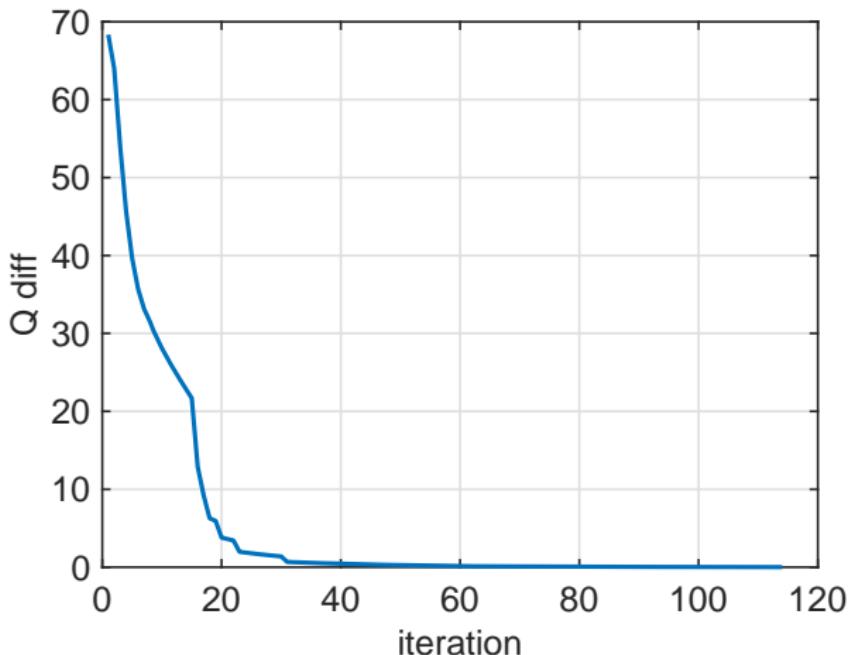
$$\{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{400}\}$$

■ 有限动作集

- 在 $[-10, 10]$ 范围选择 3 个 $u \in \{\mathcal{A}_1 = -10, \mathcal{A}_2 = 0, \mathcal{A}_3 = 10\}$ 作为代表性的动作

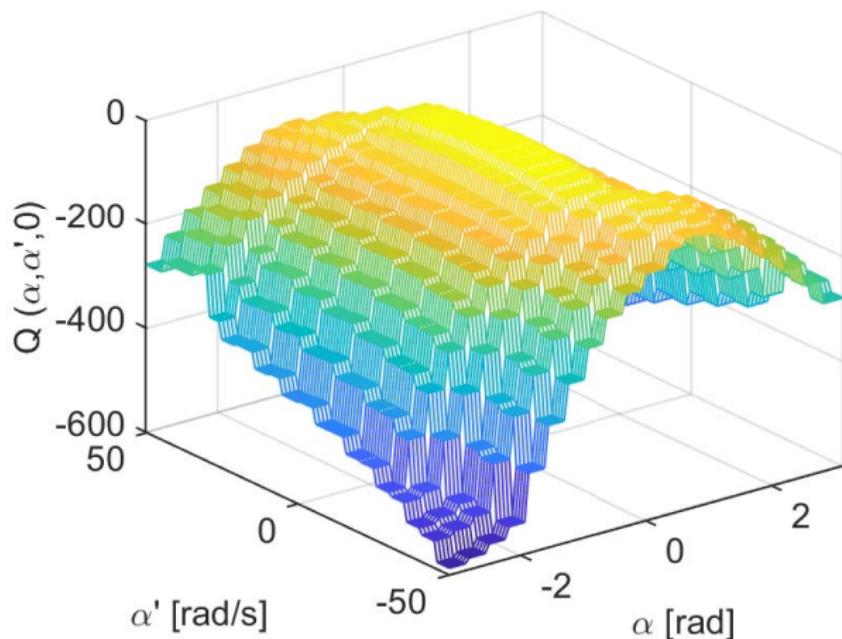
■ 离散化价值迭代学习过程

- 横坐标: 迭代次数
- 纵坐标: 相邻两次迭代 Q 值的变化 $\max_{k,l} |Q_{k,l}^{(i)} - Q_{k,l}^{(i-1)}|$

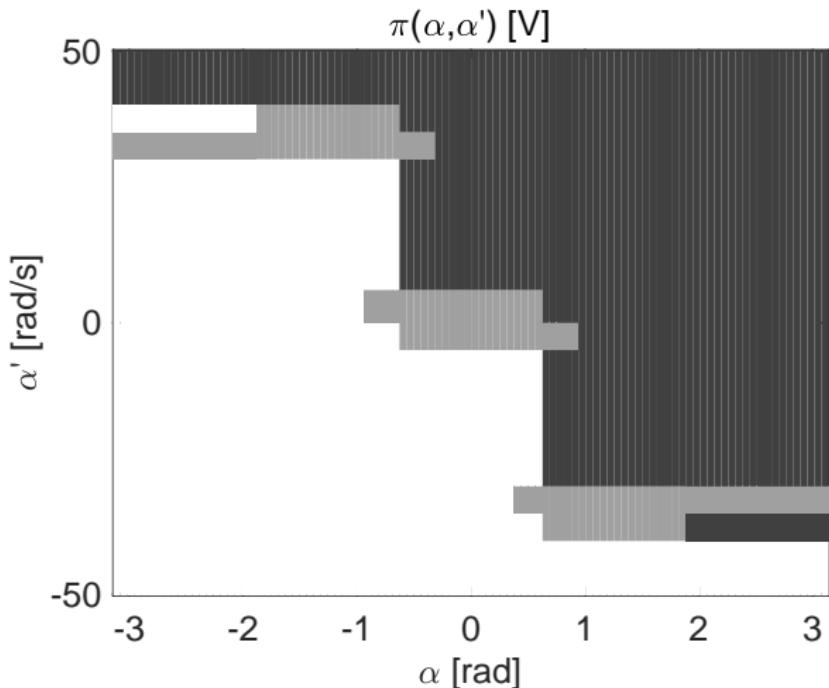


■ 收敛后 $a = 0$ 下的 Q 函数形状

$$Q(s, 0) = Q_{k,l}, \quad s \in \mathcal{S}_k, \mathcal{A}_l = 0$$

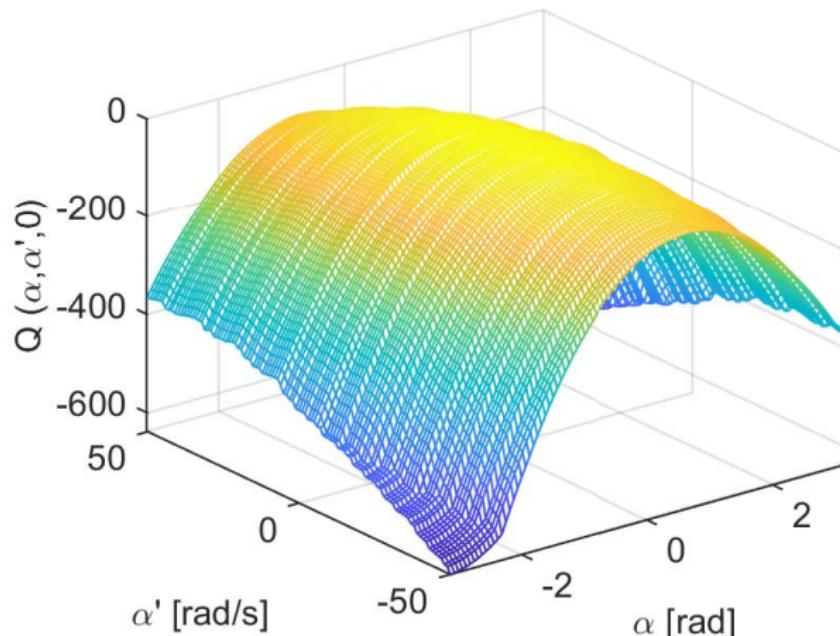


- 收敛后策略的动作分布：白色代表 $a = 10$, 灰色代表 $a = 0$, 黑色代表 $a = -10$

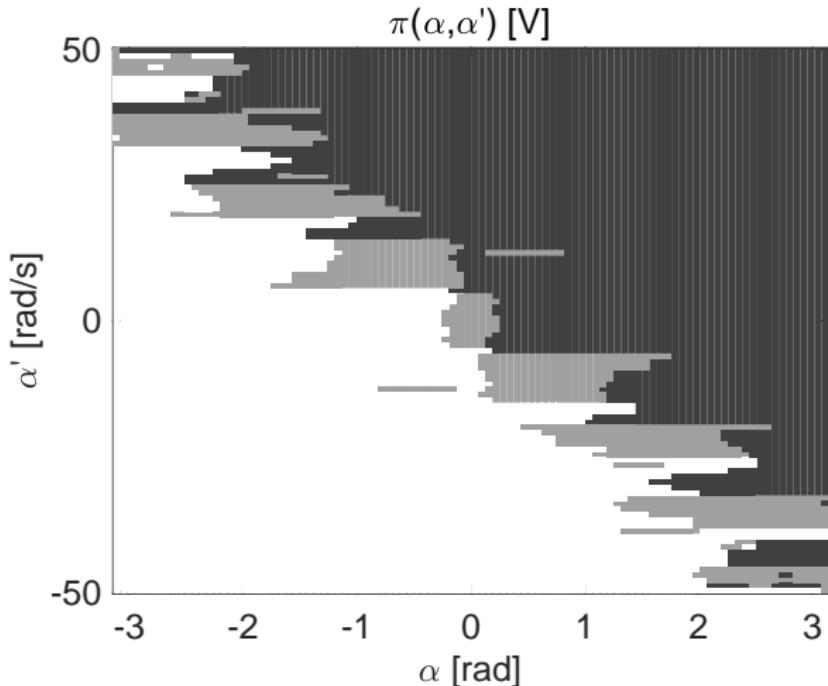


举例：直流电机 (2)

- 提高离散化的分辨率
 - 在角度和角速度范围分别均匀划分成 100 份
 - 价值迭代收敛后 $a = 0$ 下的 Q 函数形状

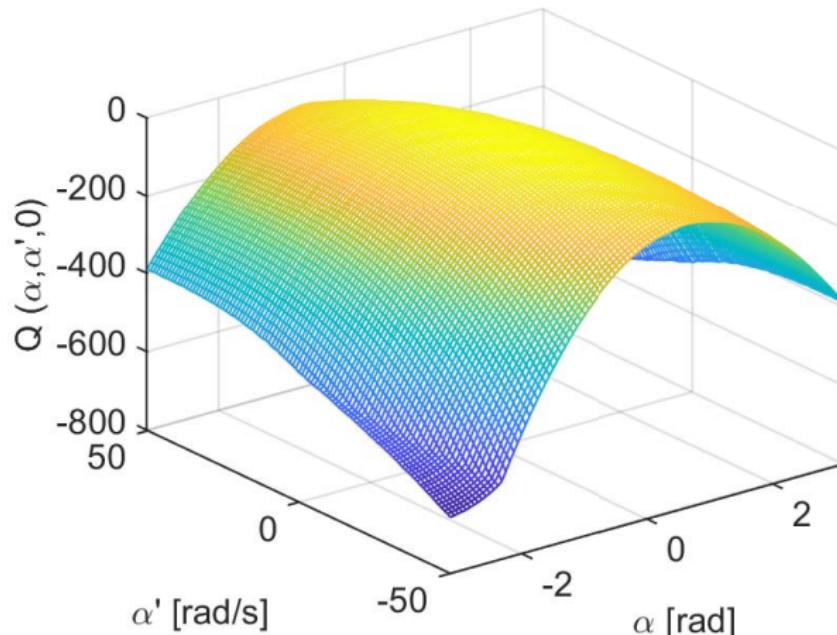


- 收敛后策略的动作分布：白色代表 $a = 10$, 灰色代表 $a = 0$, 黑色代表 $a = -10$

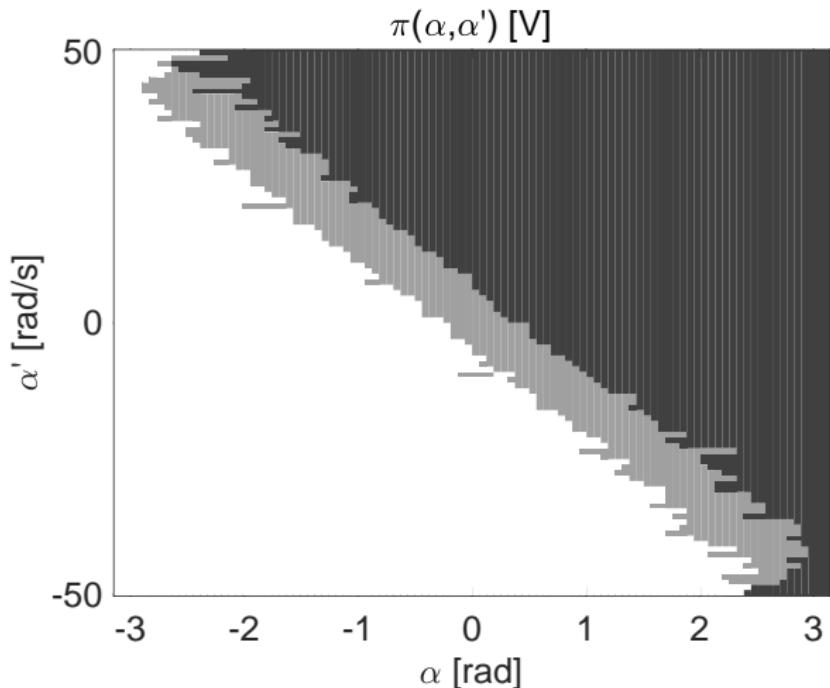


举例：直流电机 (3)

- 进一步提高离散化的分辨率
 - 在角度和角速度范围分别均匀划分成 1000 份
 - 价值迭代收敛后 $a = 0$ 下的 Q 函数形状



- 收敛后策略的动作分布：白色代表 $a = 10$, 灰色代表 $a = 0$, 黑色代表 $a = -10$



- 离散化的价值迭代依然具有 收敛性
- 离散化程度越高，结果越精确
- 但是 存储空间和计算量随离散集数量增加而增加
 - 直流电机三个实验中离散化 Q 值个数分别是：
 $20^2 \times 3 = 1200$, $100^2 \times 3 = 30000$, $1000^2 \times 3 = 3000000$
- 算法复杂度随维数呈指数增长， 维数灾的困境

Fitted Q Iteration

- 我们可以在价值迭代中使用用更高效的逼近器, e.g. coarse coding, RBF, NN, etc
- 定义 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w})$
- 和逼近器 最佳匹配映射 (best fit mapping) 算子 Π :

$$\hat{Q}(\mathbf{w}^*) = \Pi Q$$

$$s.t. \quad \mathbf{w}^* = \arg \min_{\mathbf{w}} \mathbb{E}[(Q(s, a) - \hat{Q}(s, a, \mathbf{w}))^2]$$

- 这样就可以在 \hat{Q} 代入到价值迭代算子后, 将结果重新映射回逼近器空间:

$$\hat{Q} \leftarrow \Pi \mathcal{T}(\hat{Q})$$

- FQI 流程:

- 1 初始化逼近器权重 \mathbf{w}
- 2 将当前 $Q(s, a, \mathbf{w})$ 代入 $\Pi\mathcal{T}$ 后, 得到一组新的权重 \mathbf{w}'

$$\hat{Q}(s, a, \mathbf{w}') = \Pi\left(\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} \hat{Q}(s', a', \mathbf{w})\right)$$

- 3 更新 $\mathbf{w} \leftarrow \mathbf{w}'$
- 4 返回步骤 2

- 状态空间可以是连续的, 也可以是离散的
- 动作空间依然要求是 离散, 有限的 (max 操作)

- 在动态规划中已经证明 \mathcal{T} 在无穷范数上是 γ -收缩算子

$$\|\mathcal{T}(Q_1) - \mathcal{T}(Q_2)\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$$

- 如果逼近器的映射算子 Π 在无穷范数上是收缩算子

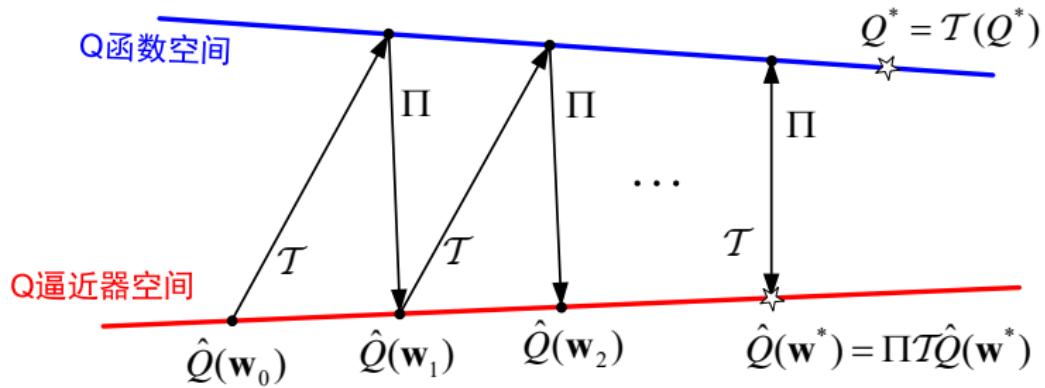
$$\|\Pi(Q_1) - \Pi(Q_2)\|_\infty \leq \|Q_1 - Q_2\|_\infty$$

- 那么 $\Pi\mathcal{T}$ 在无穷范数上依然是 γ -收缩算子
- 使用这种逼近器 FQI 能够 收敛
- 但收敛的结果不一定等于最优 Q 函数

$$\hat{Q}(s, a, \mathbf{w}^*) \neq Q^*(s, a)$$

(由于逼近误差的存在)

■ FQI 收敛示意



- 1: 定义特征向量 \mathbf{x} , 和线性 Q 逼近器 $\hat{Q}(s, a, \mathbf{w}_0) = \mathbf{x}^T \mathbf{w}_0$, 初始化权重 \mathbf{w}_0 , $i = 0$
- 2: 采样一组数据 $\{(s_k, a_k, r_{k+1}, s_{k+1})\}, k = 1, \dots, K$
- 3: **repeat**
- 4: **for all** $k = 1, \dots, K$ **do**
- 5: 计算 $q_k = r_{k+1} + \gamma \max_{a'} \hat{Q}(s_{k+1}, a', \mathbf{w}_i)$
- 6: **end for**
- 7: 更新 $\mathbf{w}_{i+1} = \arg \min_{\mathbf{w}} \frac{1}{K} \sum_{k=1}^K (q_k - \mathbf{x}(s_k, a_k)^T \mathbf{w})^2$
- 8: $i \leftarrow i + 1$
- 9: **until** \mathbf{w}_i 收敛或迭代一定次数

- 1: 定义特征向量 \mathbf{x} , 和线性 Q 逼近器 $\hat{Q}(s, a, \mathbf{w}_0) = \mathbf{x}^T \mathbf{w}_0$, 初始化权重 \mathbf{w}_0 , $i = 0$
- 2: 采样一组数据 $\{(s_k, a_k, r_{k+1}, s_{k+1})\}, k = 1, \dots, K$
- 3: **repeat**
- 4: **for all** $k = 1, \dots, K$ **do**
- 5: 计算 $q_k = r_{k+1} + \gamma \max_{a'} \hat{Q}(s_{k+1}, a', \mathbf{w}_i)$
- 6: **end for**
- 7: 更新 $\mathbf{w}_{i+1} = \arg \min_{\mathbf{w}} \frac{1}{K} \sum_{k=1}^K (q_k - \mathbf{x}(s_k, a_k)^T \mathbf{w})^2$
- 8: $i \leftarrow i + 1$
- 9: **until** \mathbf{w}_i 收敛或迭代一定次数

- 用 样本 (r_{k+1}, s_{k+1}) 近似奖励和转移函数 \mathcal{R}, \mathcal{P}
- 用 数据集 $\{(s_k, a_k)\}$ 近似状态和动作空间
- 如果事先能够采样到这样一组样本, 就可以 无模型的 强化学习

策略迭代 + 最小二乘

- 同样可以把 linear approximation+least-squared 用在策略评估上
- 回顾：有限 MDPs 策略评估：
 - MC: 为了让价值尽可能等于回报 $V_\pi(s_t) \approx G_t$
 - TD: 为了让贝尔曼期望方程等号成立
$$V_\pi(s_t) \approx r_{t+1} + \gamma V_\pi(s_{t+1})$$

- 同样可以把 linear approximation+least-squared 用在策略评估上
- 回顾：有限 MDPs 策略评估：
 - MC: 为了让价值尽可能等于回报 $V_\pi(s_t) \approx G_t$
 - TD: 为了让贝尔曼期望方程等号成立

$$V_\pi(s_t) \approx r_{t+1} + \gamma V_\pi(s_{t+1})$$
- 对大规模 MDPs, 定义策略 π 的线性价值逼近器

$$\hat{V}_\pi(s, \mathbf{w}) = \mathbf{x}^T(s)\mathbf{w}$$
- 让智能体执行 π 然后采样数据集
 - LSMC: $\{(s_t, G_t)\}, \tau \sim \pi$
 - LSTD: $\{(s_t, a_t, r_{t+1}, s_{t+1})\}, a_t \sim \pi(s_t)$

线性最小二乘预测算法



$$LSMC \quad \min_{\mathbf{w}} \sum_{t=1}^T (G_t - \mathbf{x}^T(s_t)\mathbf{w})^2$$

$$\Rightarrow \mathbf{w} = \left(\sum_{t=1}^T \mathbf{x}(s_t) \mathbf{x}(s_t)^T \right)^{-1} \sum_{t=1}^T \mathbf{x}(s_t) G_t$$

$$LSTD \quad \min_{\mathbf{w}} \sum_{t=1}^T (r_{t+1} + \gamma \mathbf{x}^T(s_{t+1})\mathbf{w} - \mathbf{x}^T(s_t)\mathbf{w})^2$$

$$\Rightarrow \mathbf{w} = \left(\sum_{t=1}^T \mathbf{x}(s_t) (\mathbf{x}(s_t) - \gamma \mathbf{x}(s_{t+1}))^T \right)^{-1} \sum_{t=1}^T \mathbf{x}(s_t) r_{t+1}$$

$$\min_{\mathbf{w}} \sum_{t=1}^T (r_{t+1} + \gamma \mathbf{x}^T(s_{t+1}) \mathbf{w} - \mathbf{x}^T(s_t) \mathbf{w})^2$$

$$\Rightarrow \mathbf{R} = \left[\begin{array}{c|c} & \\ r_{t+1} & \\ & \end{array} \right], \mathbf{X}' = \left[\begin{array}{c|c} & \\ \mathbf{x}^T(s_{t+1}) & \\ & \end{array} \right], \mathbf{X} = \left[\begin{array}{c|c} & \\ \mathbf{x}^T(s_t) & \\ & \end{array} \right]$$

$$\begin{aligned} & \Rightarrow 0 \approx \mathbf{R} + \gamma \mathbf{X}' \mathbf{w} - \mathbf{X} \mathbf{w} \\ & (\mathbf{X} - \gamma \mathbf{X}') \mathbf{w} \approx \mathbf{R} \\ & \mathbf{X}^T (\mathbf{X} - \gamma \mathbf{X}') \mathbf{w} = \mathbf{X}^T \mathbf{R} \\ & \mathbf{w} = (\mathbf{X}^T (\mathbf{X} - \gamma \mathbf{X}'))^{-1} \mathbf{X}^T \mathbf{R} \end{aligned}$$

- 策略估计 + 策略提升 \rightarrow 策略迭代
- 为了避免对模型的依赖, 对策略 π 评估 Q 函数

- 策略估计 + 策略提升 \rightarrow 策略迭代
- 为了避免对模型的依赖, 对策略 π 评估 Q 函数
- 设计线性 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w}) = \mathbf{x}^T(s, a)\mathbf{w}$, 代入贝尔曼期望方程

$$\hat{Q}(s, a, \mathbf{w}) \approx \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \sum_a \pi(a'|s') \hat{Q}(s', a', \mathbf{w})$$

- 利用样本 $\{(s_t, a_t, r_{t+1}, s_{t+1})\}$ 和最小二乘确定最佳匹配权重

$$LSTD-Q \quad \min_{\mathbf{w}} \sum_{t=1}^T \left(r_{t+1} + \gamma \sum_{a'} \pi(a'|s_{t+1}) \mathbf{x}^T(s_{t+1}, a') \mathbf{w} - \mathbf{x}^T(s_t, a_t) \mathbf{w} \right)^2$$

$$\mathbf{w} = \left(\sum_{t=1}^T \mathbf{x}(s_t, a_t) \left[\mathbf{x}(s_t, a_t) - \gamma \sum_{a'} \pi(a'|s_{t+1}) \mathbf{x}(s_{t+1}, a') \right]^T \right)^{-1} \sum_{t=1}^T \mathbf{x}(s_t, a_t) r_{t+1}$$

- LSTD-Q 中样本 $\{(s_t, a_t, r_{t+1}, s_{t+1})\}$ 的 a_t 是不是根据策略 π 采样, $a_t \sim \pi(s_t)$?

- LSTD-Q 中样本 $\{(s_t, a_t, r_{t+1}, s_{t+1})\}$ 的 a_t 是不是根据策略 π 采样, $a_t \sim \pi(s_t)$?
 - 样本是为了求解 Q 函数, $Q(s, a)$ 输入的是状态和动作
 - 样本 $\{(s_t, a_t)\}$ 应当尽可能地全面代表状态空间和动作空间, 计算出的 $\hat{Q}(s_t, a_t)$ 才能足够逼近真实 $Q_\pi(s, a)$

- LSTD-Q 中样本 $\{(s_t, a_t, r_{t+1}, s_{t+1})\}$ 的 a_t 是不是根据策略 π 采样, $a_t \sim \pi(s_t)$?
 - 样本是为了求解 Q 函数, $Q(s, a)$ 输入的是状态和动作
 - 样本 $\{(s_t, a_t)\}$ 应当尽可能地全面代表状态空间和动作空间, 计算出的 $\hat{Q}(s_t, a_t)$ 才能足够逼近真实 $Q_\pi(s, a)$
- 如果 π 是确定策略:
 - a_t 完全根据 π 采样, $\hat{Q}(s_t, a_t) = \hat{Q}(s_t, \pi(s_t))$ 会变成只和状态有关的函数, 计算的结果和 $Q_\pi(s, a)$ 误差很大
 - 这样就需要具有探索性的动作, 提高逼近器的泛化能力, e.g. $a_t \sim \epsilon\text{-greedy}(\pi)$

- LSTD-Q 中样本 $\{(s_t, a_t, r_{t+1}, s_{t+1})\}$ 的 a_t 是不是根据策略 π 采样, $a_t \sim \pi(s_t)$?
 - 样本是为了求解 Q 函数, $Q(s, a)$ 输入的是状态和动作
 - 样本 $\{(s_t, a_t)\}$ 应当尽可能地 全面代表状态空间和动作空间, 计算出的 $\hat{Q}(s_t, a_t)$ 才能足够逼近真实 $Q_\pi(s, a)$
- 如果 π 是 确定策略:
 - a_t 完全根据 π 采样, $\hat{Q}(s_t, a_t) = \hat{Q}(s_t, \pi(s_t))$ 会变成只和状态有关的函数, 计算的结果和 $Q_\pi(s, a)$ 误差很大
 - 这样就需要具有探索性的动作, 提高逼近器的泛化能力, e.g. $a_t \sim \epsilon\text{-greedy}(\pi)$
- 如果 π 是 随机策略, 并且 $\pi(a|s) > 0, \forall s, a$:
 - 可以根据 π 采样 $a_t, a_t \sim \pi(s_t)$
 - 否则还是需要探索的动作

- 1: 给定策略 π_0 , 定义特征向量 $\mathbf{x}(s, a)$, 初始化 $i = 0$
- 2: **repeat**
- 3: 采样一组数据集 $\{(s_t, a_t, r_{t+1}, s_{t+1})\}$
- 4: (策略评估:) 针对策略 π_i 使用 LSTD-Q 计算权重 \mathbf{w}_i , 得到策略的线性 Q 函数逼近器 $\hat{Q}_i(s, a, \mathbf{w}_i) = \mathbf{x}^T(s, a)\mathbf{w}_i$
- 5: (策略提升:) 提取贪心策略 $\pi_{i+1}(s) = \arg \max_a \hat{Q}_i(s, a, \mathbf{w}_i)$
- 6: $i \leftarrow i + 1$
- 7: **until** π_i 收敛或迭代一定次数

- 1: 给定策略 π_0 , 定义特征向量 $\mathbf{x}(s, a)$, 初始化 $i = 0$
- 2: **repeat**
- 3: 采样一组数据集 $\{(s_t, a_t, r_{t+1}, s_{t+1})\}$
- 4: (策略评估:) 针对策略 π_i 使用 LSTD-Q 计算权重 \mathbf{w}_i , 得到策略的线性 Q 函数逼近器 $\hat{Q}_i(s, a, \mathbf{w}_i) = \mathbf{x}^T(s, a)\mathbf{w}_i$
- 5: (策略提升:) 提取贪心策略 $\pi_{i+1}(s) = \arg \max_a \hat{Q}_i(s, a, \mathbf{w}_i)$
- 6: $i \leftarrow i + 1$
- 7: **until** π_i 收敛或迭代一定次数

- 如果数据集有足够的 **多样性**, 每次迭代可以使用同一组数据

举例：直流电机

- 二维状态，一维动作的连续状态转移函数

$$s_{t+1} = \mathcal{P}(s_t, a_t) = \begin{bmatrix} 1 & 0.0049 \\ 0 & 0.9540 \end{bmatrix} s_t + \begin{bmatrix} 0.0021 \\ 0.8505 \end{bmatrix} a_t$$

- $s_t(1) = \alpha \in [-\pi, \pi]$ rad 电机角度
- $s_t(2) = \dot{\alpha} \in [-16\pi, 16\pi]$ rad/s 角速度
- $a_t \in [-10, 10]$ V 输入电压

- 奖励函数

$$r_{t+1} = \mathcal{R}(s_t, a_t) = -c_1 s_t(1)^2 - c_2 s_t(2)^2 - c_3 a_t^2,$$

$$c_1 = 5, c_2 = 0.01, c_3 = 0.01$$

- 折扣因子 $\gamma = 0.95$

举例：直流电机问题

- 定义 状态 - 高斯核函数 $\phi_i(s)$: 中心点 $c_i = [c_{i,1}, c_{i,2}]^T$, 协方差矩阵 $B_i = \begin{bmatrix} \sigma_{i,1}^2 & 0 \\ 0 & \sigma_{i,2}^2 \end{bmatrix}$
 - $c_{i,1}$ 在角度 $[-\pi, \pi]$ 范围均匀选取 9 个点
 - $c_{i,2}$ 在角速度 $[-16\pi, 16\pi]$ 范围均匀选取 9 个点
 - $\sigma_{i,1} = \frac{2\pi}{8}$
 - $\sigma_{i,2} = \frac{32\pi}{8}$

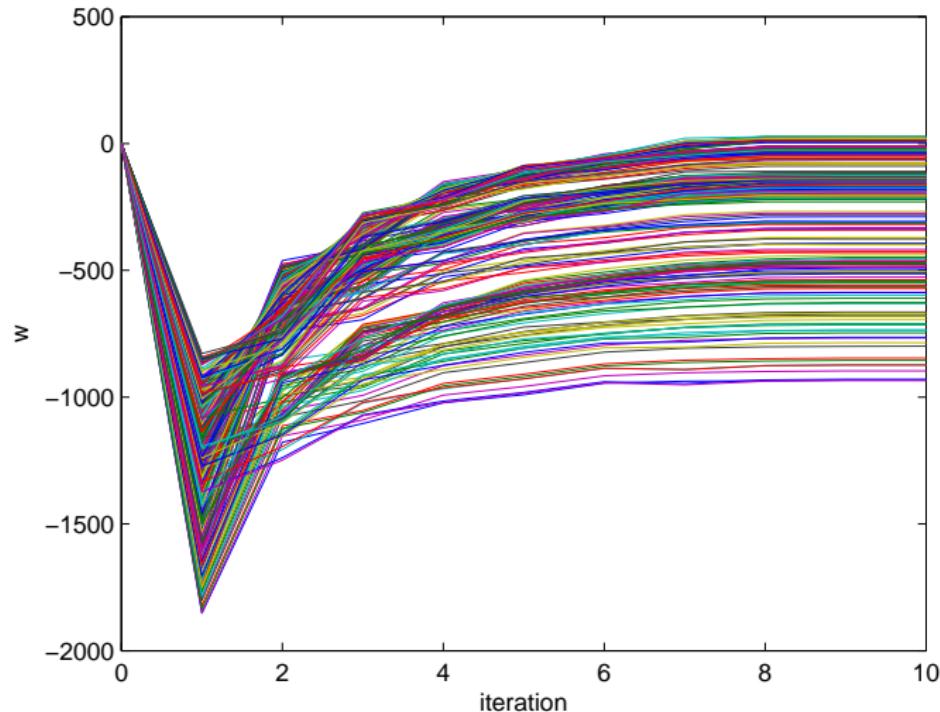
- 动作空间离散化成 $\{\mathcal{A}_1 = -10, \mathcal{A}_2 = 0, \mathcal{A}_3 = 10\}$
- 状态 - 动作特征向量 \mathbf{x} 由 $\{\phi_i\}$ 沿离散动作堆叠而成

$\mathbf{x}(s, a) =$

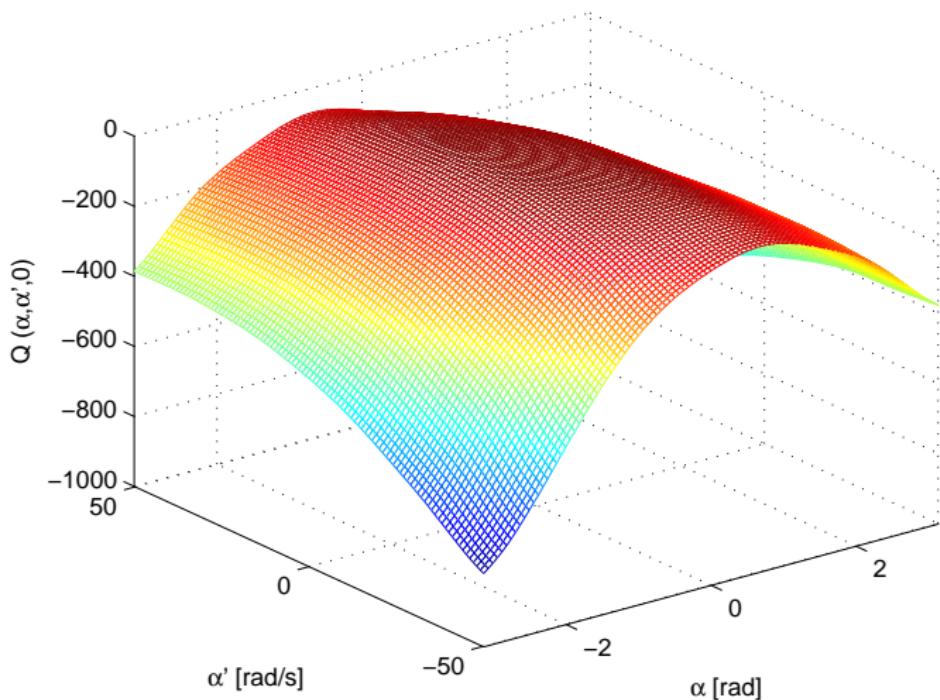
$$[[\phi_1(s), \phi_2(s), \dots] \mathcal{I}(a = \mathcal{A}_1), [\phi_1(s), \dots] \mathcal{I}(a = \mathcal{A}_2), [\phi_1(s), \dots] \mathcal{I}(a = \mathcal{A}_3)]^T$$

- 特征向量元素一共有 $9 * 9 * 3 = 243$ 个
- 随机选取 7500 个样本

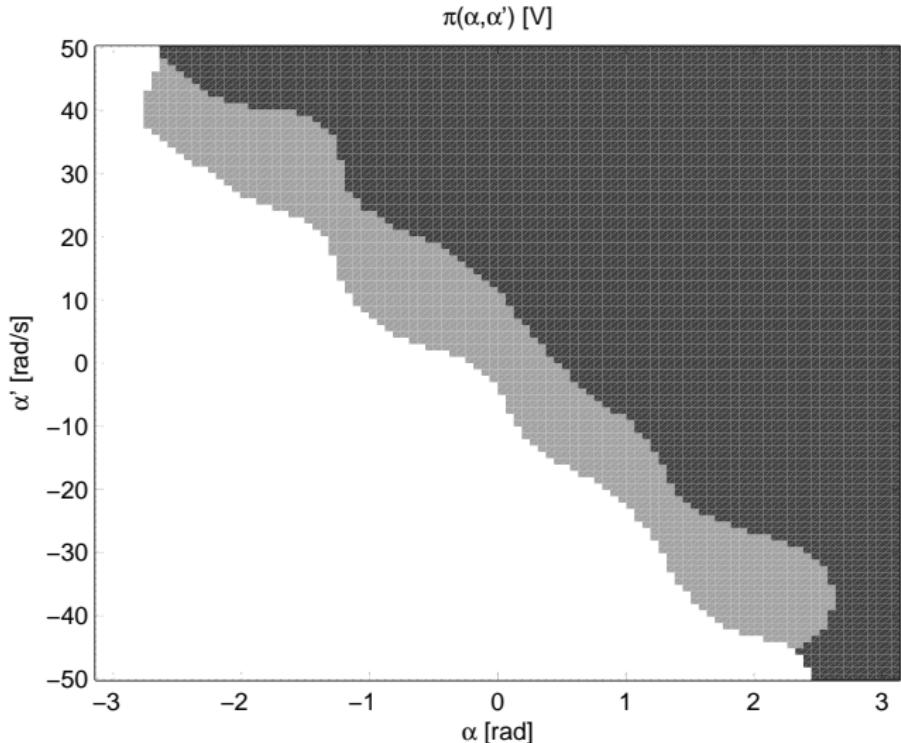
■ LSPI 算法中线性逼近器权重变化曲线



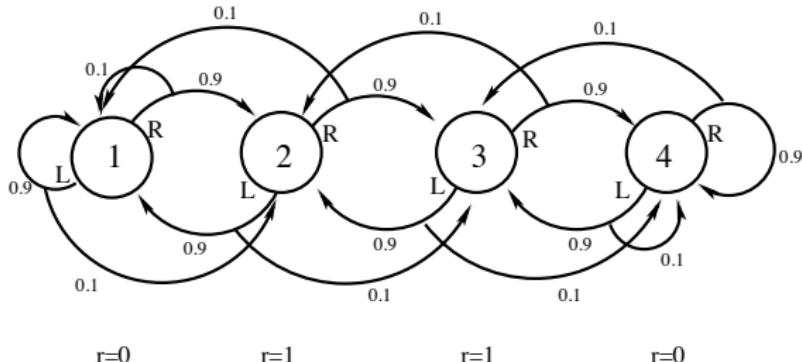
■ 收敛后 $a = 0$ 下 Q 函数形状



■ 收敛后策略的动作分布



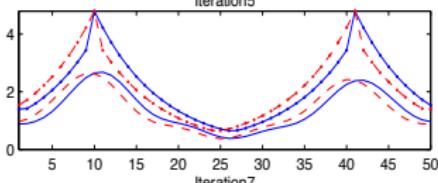
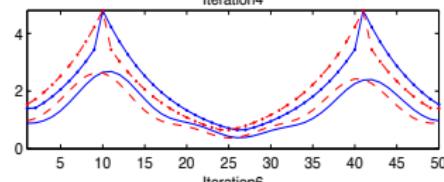
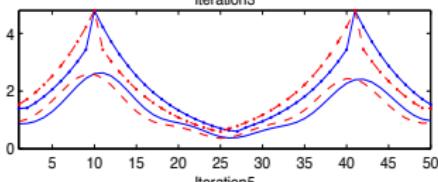
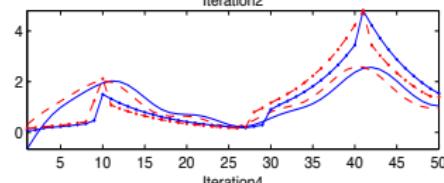
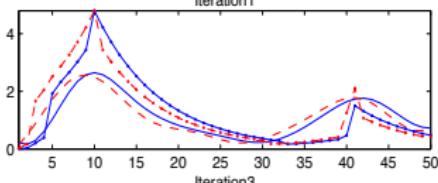
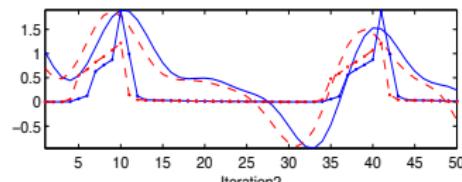
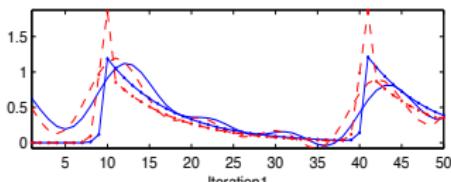
举例：链式 MDP 行走问题



- 考虑有 50 个状态的链式 MDP 问题
- 在状态 10 和 41 上获得奖励 +1, 其它状态奖励为 0
- 最优策略: R (1-9), L(10-25), R(26-41), L(42-50)
- 特征: 为每个动作定义 10 个均匀分布的高斯 RBF ($\sigma = 4$)
- 样本: 由随机策略产生的 10000 步数据

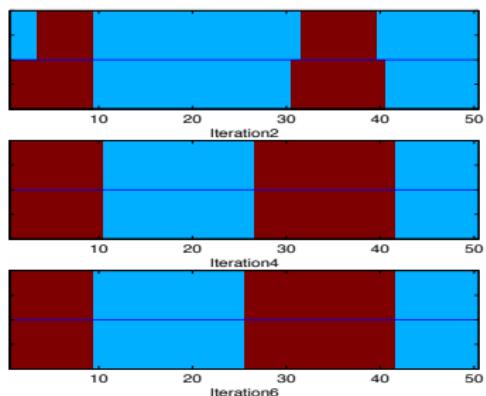
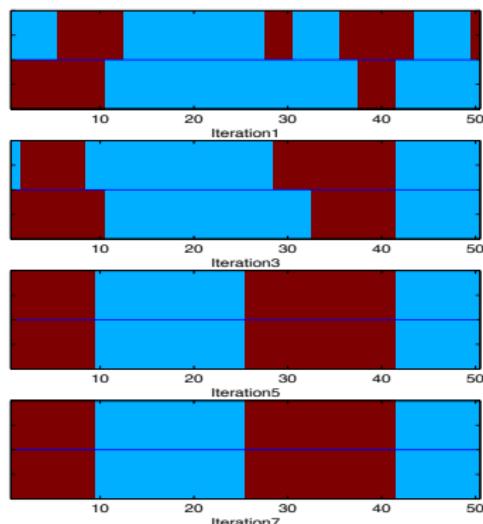
■ LSPI 算法中 Q 函数变化

- The state-action value function of the policy being evaluated in each iteration (LSPI approximation - solid lines; exact values - dotted lines).



■ LSPI 算法中策略变化

- The improved policy after each iteration (R action - dark/red shade; L action - light/blue shade; LSPI - top stripe; exact - bottom stripe).



- 共同的优点: 和 收缩逼近器 结合后迭代依然是收敛的
- 共同的缺点: 每次迭代的计算量巨大
 - 线性最小二乘法求样本特征矩阵的逆
- 通常 PI 的迭代次数要比 VI 迭代次数少, 因此 approximate PI 整体计算量比 approximate VI 要小

预测学习 + 随机梯度下降法

- 随机梯度下降法每次更新只计算一个样本

$$\Delta \mathbf{w} = \alpha (V_\pi(s_t) - \hat{V}(s_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w})$$

- 蒙特卡洛 预测学习中的回报 G_t 是 $V_\pi(s_t)$ 的 无偏估计
- 根据策略生成的轨迹使用梯度下降更新逼近器的权重

$$\Delta \mathbf{w} = \alpha (G_t - \hat{V}(s_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w})$$

- 1: 给定策略 π , 定义价值逼近器 $\hat{V}(s, \mathbf{w})$, 初始化 \mathbf{w}
- 2: **repeat** {对每个 episode}
- 3: 根据 π 采样轨迹 $\{s_0, a_0, r_1, s_1, \dots, s_T\}$
- 4: **repeat** {对 episode 中每个首次访问的 s_t }
- 5: 计算回报 $G_t = r_{t+1} + \gamma r_{t+2} + \dots$
- 6: 计算更新量 $\Delta \mathbf{w} = \alpha(G_t - \hat{V}(s_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w})$
- 7: 更新权重 $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
- 8: **until**
- 9: **until**

```
1: 给定策略  $\pi$ , 定义价值逼近器  $\hat{V}(s, \mathbf{w})$ , 初始化  $\mathbf{w}$ 
2: repeat {对每个 episode}
3:   根据  $\pi$  采样轨迹  $\{s_0, a_0, r_1, s_1, \dots, s_T\}$ 
4:   repeat {对 episode 中每个首次访问的  $s_t$ }
5:     计算回报  $G_t = r_{t+1} + \gamma r_{t+2} + \dots$ 
6:     计算更新量  $\Delta \mathbf{w} = \alpha(G_t - \hat{V}(s_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w})$ 
7:     更新权重  $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$ 
8:   until
9: until
```

- 线性和非线性逼近器都适用

- 时间差分方法使用 **自举和采样** 近似真实的 V_π
- 每观测量一次数据 $(s_t, a_t, r_{t+1}, s_{t+1})$, 更新一次 V

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

- 目标是 $r_{t+1} + \gamma V(s_{t+1})$, 是真实 V_π 的 **有偏** 估计

- 时间差分方法使用 **自举和采样** 近似真实的 V_π
- 每观测量一次数据 $(s_t, a_t, r_{t+1}, s_{t+1})$, 更新一次 V

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

- 目标是 $r_{t+1} + \gamma V(s_{t+1})$, 是真实 V_π 的 **有偏** 估计

-
- 使用价值逼近器, 目标变成 $r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w})$, 是真实 V_π **有偏且近似** 的估计
 - 存在 3 种近似:
 - 1 用**样本**近似期望
 - 2 用**自举**近似真实价值
 - 3 用**逼近器**近似原始函数

- 1: 给定策略 π , 定义价值逼近器 $\hat{V}(s, \mathbf{w})$, 初始化 \mathbf{w} , $s_t = s_0$,
 $t = 0$
- 2: **loop**
- 3: 选择执行动作 $a_t \sim \pi(s_t)$, 观测 r_{t+1}, s_{t+1}
- 4: 计算更新量
$$\Delta \mathbf{w} = \alpha(r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w}) - \hat{V}(s_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w})$$
- 5: 更新权重 $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
- 6: $t \leftarrow t + 1$
- 7: **end loop**

- 1: 给定策略 π , 定义价值逼近器 $\hat{V}(s, \mathbf{w})$, 初始化 \mathbf{w} , $s_t = s_0$,
 $t = 0$
- 2: **loop**
- 3: 选择执行动作 $a_t \sim \pi(s_t)$, 观测 r_{t+1}, s_{t+1}
- 4: 计算更新量
$$\Delta \mathbf{w} = \alpha(r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w}) - \hat{V}(s_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w})$$
- 5: 更新权重 $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
- 6: $t \leftarrow t + 1$
- 7: **end loop**

■ 思考: n-步 TD 学习和逼近器结合的形式是什么样的?

- 回顾：对有限 MDPs，我们提出了前向和后向 TD(λ) 算法
- 前向算法使用 λ -回报

$$\Delta V(s_t) = \alpha(G_t^\lambda - V(s_t))$$

- 后向算法利用 λ -资格迹

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$E_t(S) = \gamma E_{t-1}(S) + \mathcal{I}(S = s_t), \quad \forall S$$

$$\Delta V(S) = \alpha \delta_t E_t(S), \quad \forall S$$

- 两者在更新量上是等价的，但是后向算法更适合在线学习

基于逼近器的前向 TD(λ)

- λ -回报 同样是真实 V_π 的有偏估计
- 可以根据策略 π 的轨迹对逼近器进行 前向 TD(λ) 更新

$$\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{V}(s_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w})$$

基于逼近器的后向 TD(λ)

- 基于逼近器的 后向 TD(λ) 更新: 在每次获得观测量 $(s_t, a_t, r_{t+1}, s_{t+1})$ 后

$$\delta_t = r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w}) - \hat{V}(s_t, \mathbf{w})$$

$$e_t = \gamma \lambda e_{t-1} + \nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha \delta_t e_t$$

- 与有限 MDPs 为每个 (s, a) 定义一份资格迹不同, 这里只有一个资格迹向量 e_t , 存储的是累加逼近器梯度
 $\nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w})$

定理

当使用离线更新时，基于逼近器的前向和后向 $\text{TD}(\lambda)$ 在同一条轨迹上的更新量是相同的

$$\sum_{t=0}^T \Delta \mathbf{w}_t^{TD} = \sum_{t=0}^T \Delta \mathbf{w}_t^\lambda$$

- 离线更新指的是对权重的更新是在轨迹结束后进行的
- $\Delta \mathbf{w}_t^{TD}$ 代表后向 $\text{TD}(\lambda)$ 在 t 时刻对权重的更新量:
$$\Delta \mathbf{w}_t^{TD} = \alpha \delta_t e_t$$
- $\Delta \mathbf{w}_t^\lambda$ 代表前向 $\text{TD}(\lambda)$ 在 t 时刻对权重的更新量:
$$\Delta \mathbf{w}_t^\lambda = \alpha(G_t^\lambda - \hat{V}(s_t)) \nabla_{\mathbf{w}} \hat{V}(s_t)$$

证明

- 对后向算法，轨迹上的资格迹等于

$$e_t = \sum_{k=0}^t (\gamma \lambda)^{t-k} \nabla_{\mathbf{w}} \hat{V}_k$$

- 整个后向算法的更新量等于

$$\sum_{t=0}^T \Delta \mathbf{w}_t^{TD} = \alpha \sum_{t=0}^T \nabla_{\mathbf{w}} \hat{V}_t \sum_{k=t}^T (\gamma \lambda)^{k-t} \delta_k$$

- 前向算法每一时刻的更新量等于

$$\Delta \mathbf{w}_t^\lambda = \alpha \sum_{k=t}^T (\gamma \lambda)^{k-t} \delta_k \nabla_{\mathbf{w}} \hat{V}_t$$

- 所以

$$\sum_{t=0}^T \Delta \mathbf{w}_t^{TD} = \sum_{t=0}^T \Delta \mathbf{w}_t^\lambda$$

- 1: 给定策略 π , 定义价值逼近器 $\hat{V}(s, \mathbf{w})$, 初始化 \mathbf{w} , $s_t = s_0$,
 $e_t = \mathbf{0}$, $t = 0$
- 2: **loop**
- 3: 选择执行动作 $a_t \sim \pi(s_t)$, 观测 r_{t+1}, s_{t+1}
- 4: 计算 TD 误差 $\delta_t = r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w}) - \hat{V}(s_t, \mathbf{w})$
- 5: 更新资格迹 $e_t = \gamma \lambda e_{t-1} + \nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w})$
- 6: 更新权重 $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta_t e_t$
- 7: $t \leftarrow t + 1$
- 8: **end loop**

收敛分析

- 定义策略的真实 V_π 和价值逼近器 $\hat{V}(s, \mathbf{w})$ 之间的 **均方误差**

$$MSE(\mathbf{w}) = \sum_{s \in \mathcal{S}} d(s) (V_\pi(s) - \hat{V}(s, \mathbf{w}))^2$$

其中 $d(s)$ 是智能体在策略 π 下的状态分布

- 使用 **线性逼近器** $\hat{V}(s, \mathbf{w}) = \mathbf{x}^T(s)\mathbf{w}$ 时, $TD(\lambda)$ 策略评估能够收敛 $\mathbf{w} \rightarrow \mathbf{w}_{TD}$, 并且 **收敛误差和最小均方误差** 之间满足¹

$$MSE(\mathbf{w}_{TD(\lambda)}) = \frac{1 - \lambda\alpha}{1 - \alpha} \min_{\mathbf{w}} \sum_{s \in \mathcal{S}} d(s) (V_\pi(s) - \hat{V}(s, \mathbf{w}))^2$$

¹ J. N. Tsitsiklis and B. Van Roy, An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5): 674–690, 1997.

- MC 可以看成 $\text{TD}(\lambda)$ 在 $\lambda = 1$ 的特例，它的收敛误差是

$$MSE(\mathbf{w}_{MC}) = \min_{\mathbf{w}} \sum_{s \in \mathcal{S}} d(s) (V_\pi(s) - \hat{V}(s, \mathbf{w}))^2$$

- $\text{TD}(0)$ 是 $\text{TD}(\lambda)$ 在 $\lambda = 0$ 的特例，它的收敛误差是

$$MSE(\mathbf{w}_{TD}) = \min_{\mathbf{w}} \frac{1}{1-\alpha} \sum_{s \in \mathcal{S}} d(s) (V_\pi(s) - \hat{V}(s, \mathbf{w}))^2$$

- MC 可以看成 TD(λ) 在 $\lambda = 1$ 的特例, 它的收敛误差是

$$MSE(\mathbf{w}_{MC}) = \min_{\mathbf{w}} \sum_{s \in \mathcal{S}} d(s) (V_\pi(s) - \hat{V}(s, \mathbf{w}))^2$$

- TD(0) 是 TD(λ) 在 $\lambda = 0$ 的特例, 它的收敛误差是

$$MSE(\mathbf{w}_{TD}) = \min_{\mathbf{w}} \frac{1}{1-\alpha} \sum_{s \in \mathcal{S}} d(s) (V_\pi(s) - \hat{V}(s, \mathbf{w}))^2$$

- 上述都是 线性逼近器 下的结果. 如果使用查表法表示价值函数, $MSE(\mathbf{w}_{MC})$, $MSE(\mathbf{w}_{TD})$, $MSE(\mathbf{w}_{TD(\lambda)})$ 变成了什么?

- MC 可以看成 $\text{TD}(\lambda)$ 在 $\lambda = 1$ 的特例, 它的收敛误差是

$$MSE(\mathbf{w}_{MC}) = \min_{\mathbf{w}} \sum_{s \in \mathcal{S}} d(s) (V_\pi(s) - \hat{V}(s, \mathbf{w}))^2$$

- $\text{TD}(0)$ 是 $\text{TD}(\lambda)$ 在 $\lambda = 0$ 的特例, 它的收敛误差是

$$MSE(\mathbf{w}_{TD}) = \min_{\mathbf{w}} \frac{1}{1-\alpha} \sum_{s \in \mathcal{S}} d(s) (V_\pi(s) - \hat{V}(s, \mathbf{w}))^2$$

- 上述都是 **线性逼近器** 下的结果. 如果使用查表法表示价值函数, $MSE(\mathbf{w}_{MC})$, $MSE(\mathbf{w}_{TD})$, $MSE(\mathbf{w}_{TD(\lambda)})$ 变成了什么?
- 如果使用 **非线性逼近器**, MC, $\text{TD}(0)$, $\text{TD}(\lambda)$ 的收敛性还能保证吗?

- 使用非线性逼近器, MC 和前向 TD(λ) 进行策略评估是收敛的
 - G_t 和 G_t^λ 是 V_π 的无偏估计, 属于 监督学习
(supervised learning)
- 使用非线性逼近器, TD(0) 和后向 TD(λ) 进行策略评估会发散
 - 基于自举的 TD 误差 $\delta = r + \gamma \hat{V}(s') - \hat{V}(s)$ 是 V_π 的有偏估计

- 更重要的原因: TD(0) 的更新量不属于任何二阶连续函数
的梯度

$$\begin{aligned}\delta &= r + \gamma \hat{V}' - \hat{V} \\ \Delta \mathbf{w} &= \alpha \delta \nabla_{\mathbf{w}} \hat{V}\end{aligned}$$

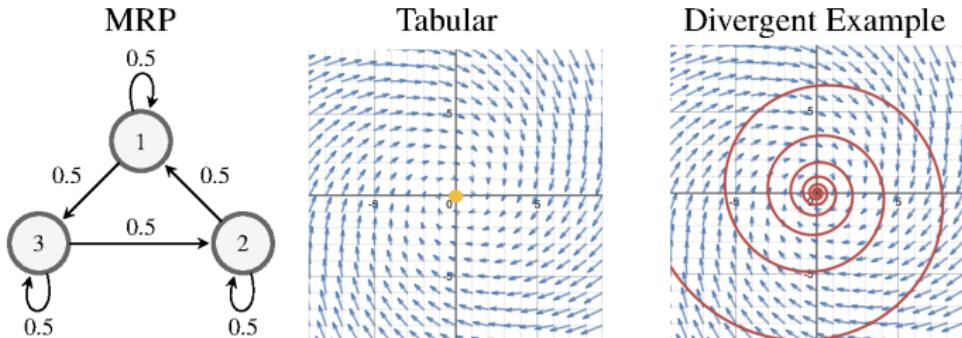
- 反证法: 假设存在连续函数 J 满足 $\partial J / \partial \mathbf{w} = \delta \nabla_{\mathbf{w}} \hat{V}$
- 对于二阶偏导

$$\left\{ \begin{array}{l} \frac{\partial^2 J}{\partial \mathbf{w}_j \partial \mathbf{w}_i} = \frac{\partial(\delta(\nabla_{\mathbf{w}} \hat{V})_i)}{\mathbf{w}_j} = (\gamma(\nabla_{\mathbf{w}} \hat{V}')_j - (\nabla_{\mathbf{w}} \hat{V})_j)(\nabla_{\mathbf{w}} \hat{V})_i + \delta(\nabla_{\mathbf{w}}^2 \hat{V})_{ji} \\ \frac{\partial^2 J}{\partial \mathbf{w}_i \partial \mathbf{w}_j} = \frac{\partial(\delta(\nabla_{\mathbf{w}} \hat{V})_j)}{\mathbf{w}_i} = (\gamma(\nabla_{\mathbf{w}} \hat{V}')_i - (\nabla_{\mathbf{w}} \hat{V})_i)(\nabla_{\mathbf{w}} \hat{V})_j + \delta(\nabla_{\mathbf{w}}^2 \hat{V})_{ij} \end{array} \right.$$

$$\Rightarrow \frac{\partial^2 J}{\partial \mathbf{w}_j \partial \mathbf{w}_i} \neq \frac{\partial^2 J}{\partial \mathbf{w}_i \partial \mathbf{w}_j}$$

- 与二阶偏导是对称的性质相矛盾, 所以不存在这样的 J

- Tsitsiklis & Van Roy (1997)² 提供了一个使用非线性逼近器的 TD 方法是发散的例子

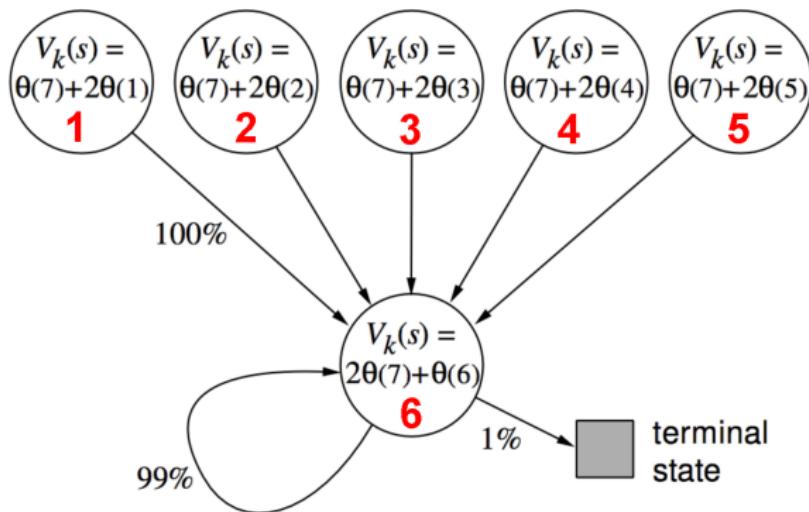


-figure from Brandfonbrener & Bruna (2020)³

- 左图的 MRP 所有奖励都是 0, 所以 $V = 0$
- 右边两图中的蓝色向量场代表价值函数更新时的动力学方向
- 如果使用查表法, 价值函数随着向量场收敛到 0 点
- 如果使用逼近器, 会沿着逼近器的函数空间移动, 可能会如右图红色螺旋线一样向外发散

² J. N. Tsitsiklis & B. Van Roy (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*.

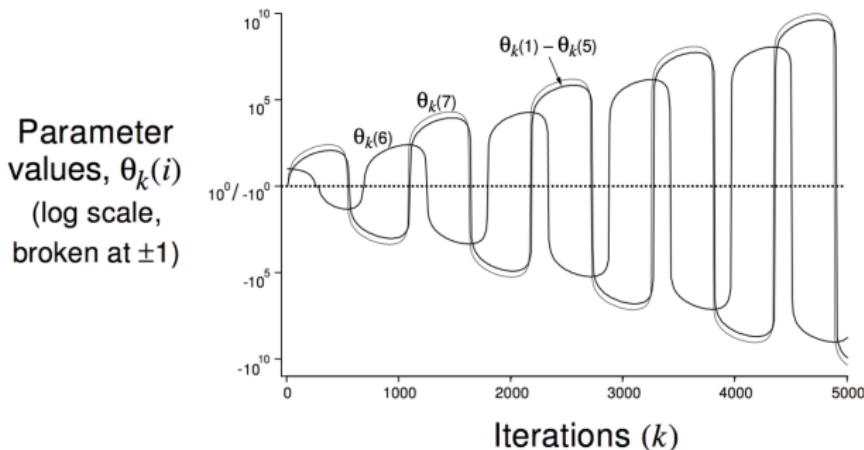
³ Brandfonbrener, D., & Bruna, J. (2019). Geometric Insights into the Convergence of Nonlinear TD Learning. *arXiv preprint arXiv:1905.12185*.



- 6 个状态，每个状态转移的奖励都是 0，因此真实的 $V(s) = 0, \forall s$

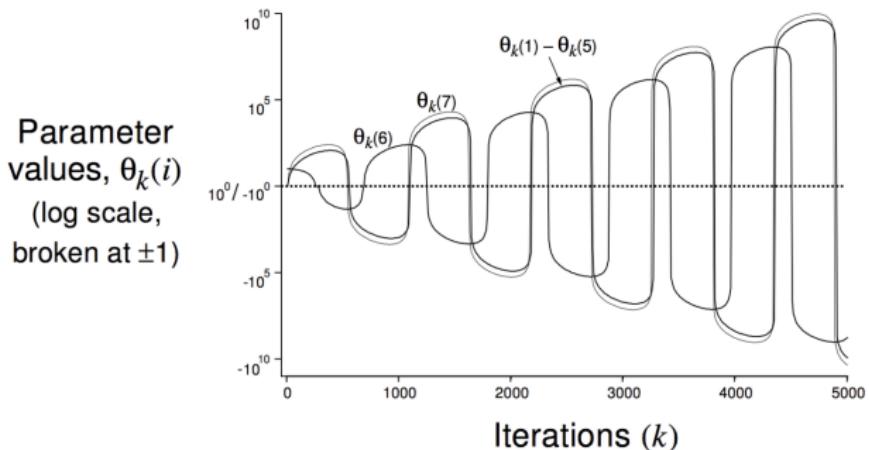
⁴Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. *Machine Learning Proceedings 1995*, pp. 30-37.

- 定义逼近器对每个状态的价值等于两个权重的线性组合， $\theta = 0$ 即可得到真实的价值
- 在使用 TD(0) 对权重学习时，用均匀的概率采样每个状态
- 从任意初始权重开始学习，权重是否一定能收敛？



$$\gamma = 0.99, \alpha = 0.01, \theta_0 = [1, 1, 1, 1, 1, 10, 1]^T$$

- 如果初始化时所有的权重都是正的，并且 θ_6 远大于其它的权重，那么 TD(0) 学习会导致权重发散
- 原因是什么？



$$\gamma = 0.99, \alpha = 0.01, \theta_0 = [1, 1, 1, 1, 1, 10, 1]^T$$

- 如果初始化时所有的权重都是正的，并且 θ_6 远大于其它的权重，那么 TD(0) 学习会导致权重发散
- 原因是什么？ 离策略学习过程
 - 训练时所有状态以相同概率采样 (样本分布)
 - 真实的轨迹中状态 6 的分布远大于其它状态 (期望分布)

在/离策略	算法	查表法	线性逼近	非线性逼近
在策略	MC	✓	✓	✓
	TD	✓	✓	✗
	前向 TD(λ)	✓	✓	✓
	后向 TD(λ)	✓	✓	✗
离策略	MC	✓	✓	✓
	TD	✓	✗	✗
	前向 TD(λ)	✓	✓	✓
	后向 TD(λ)	✓	✗	✗

在/离策略	算法	查表法	线性逼近	非线性逼近
在策略	MC	✓	✓	✓
	TD	✓	✓	✗
	前向 TD(λ)	✓	✓	✓
	后向 TD(λ)	✓	✓	✗
离策略	MC	✓	✓	✓
	TD	✓	✗	✗
	前向 TD(λ)	✓	✓	✓
	后向 TD(λ)	✓	✗	✗

- 尽管 TD 学习并不具有最好的收敛性, 但它适用于在线学习, 计算简便, 学习速率快, 而且通过其它一些技巧可以有效避免发散的问题, 所以在实际应用中 TD 算法使用最广泛

控制学习 + 随机梯度下降法

- 将前面基于逼近器的 **预测学习** 和 **策略提升** 结合, 即可实现针对大规模 MDPs 的强化学习控制算法
 - 1 定义 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w})$, 使用预测学习进行梯度下降训练
 - 2 策略提升时定义具有一定探索性的策略, e.g. ϵ -greedy($\hat{Q}(s, a, \mathbf{w})$), 避免收敛到局部最优解

- 对 $\hat{Q}(s, a, \mathbf{w})$ 进行梯度下降更新时，同样需要 目标
 - 对 MC，目标是回报 G_t

$$\Delta \mathbf{w} = \alpha(\textcolor{red}{G_t} - \hat{Q}(s_t, a_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t, \mathbf{w})$$

- 对 TD(0)，目标是 TD 目标

$$\Delta \mathbf{w} = \alpha(\textcolor{red}{r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}, \mathbf{w})} - \hat{Q}(s_t, a_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t, \mathbf{w})$$

- 对前向 TD(λ)，目标是动作 - 价值的 λ -回报

$$\Delta \mathbf{w} = \alpha(\textcolor{red}{G_t^\lambda} - \hat{Q}(s_t, a_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t, \mathbf{w})$$

- 对后向 TD(λ)，与上述等价的更新公式是

$$\delta_t = r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}, \mathbf{w}) - \hat{Q}(s_t, a_t, \mathbf{w})$$

$$e_t = \gamma \lambda e_{t-1} + \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha \delta_t e_t$$

- 1: 定义 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w})$, 初始化 \mathbf{w}
- 2: 提取出 ϵ 贪心策略 $\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$
- 3: **repeat** {对每个 episode}
- 4: 根据 π 采样轨迹 $\{s_0, a_0, r_1, s_1, \dots, s_T\}$
- 5: **repeat** {对 episode 中每个首次访问的 (s_t, a_t) }
- 6: 计算回报 $G_t = r_{t+1} + \gamma r_{t+2} + \dots$
- 7: 计算更新量 $\Delta \mathbf{w} = \alpha(G_t - \hat{Q}(s_t, a_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t, \mathbf{w})$
- 8: 更新权重 $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
- 9: **until**
- 10: 更新策略 $\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$
- 11: **until**

- 1: 定义 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w})$, 初始化 \mathbf{w}
- 2: 提取出 ϵ 贪心策略 $\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$
- 3: **repeat** {对每个 episode}
- 4: 根据 π 采样轨迹 $\{s_0, a_0, r_1, s_1, \dots, s_T\}$
- 5: **repeat** {对 episode 中每个首次访问的 (s_t, a_t) }
- 6: 计算回报 $G_t = r_{t+1} + \gamma r_{t+2} + \dots$
- 7: 计算更新量 $\Delta \mathbf{w} = \alpha(G_t - \hat{Q}(s_t, a_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t, \mathbf{w})$
- 8: 更新权重 $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
- 9: **until**
- 10: 更新策略 $\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$
- 11: **until**

- 将回报 G_t 替换成 λ -回报 G_t^λ 就是梯度下降的前向 TD(λ) 控制算法

梯度下降 Sarsa 算法

- 1: 定义 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w})$, 初始化 \mathbf{w} ,
 $\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$, $s_0 = s_0$, $t = 0$
- 2: 采样动作 $a_t \sim \pi(s_t)$ 并执行, 观测 (r_{t+1}, s_{t+1})
- 3: **loop**
- 4: 采样动作 $a_{t+1} \sim \pi(s_{t+1})$ 并执行, 观测 r_{t+2}, s_{t+2}
- 5: 根据 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ 计算 TD 误差

$$\delta_t = r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}, \mathbf{w}) - \hat{Q}(s_t, a_t, \mathbf{w})$$
- 6: 更新权重 $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta_t \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t, \mathbf{w})$
- 7: 更新策略 $\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$
- 8: $t \leftarrow t + 1$
- 9: **end loop**

梯度下降 Sarsa(λ) 算法



1: 定义 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w})$, 初始化 \mathbf{w} , $e_t = \mathbf{0}$,

$$\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w})), s_t = s_0, t = 0$$

2: 采样动作 $a_t \sim \pi(s_t)$ 并执行, 观测 (r_{t+1}, s_{t+1})

3: **loop**

4: 采样动作 $a_{t+1} \sim \pi(s_{t+1})$ 并执行, 观测 r_{t+2}, s_{t+2}

5: 根据 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ 计算 TD 误差

$$\delta_t = r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}, \mathbf{w}) - \hat{Q}(s_t, a_t, \mathbf{w})$$

6: 更新资格迹 $e_t = \gamma \lambda e_{t-1} + \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t, \mathbf{w})$

7: 更新权重 $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta_t e_t$

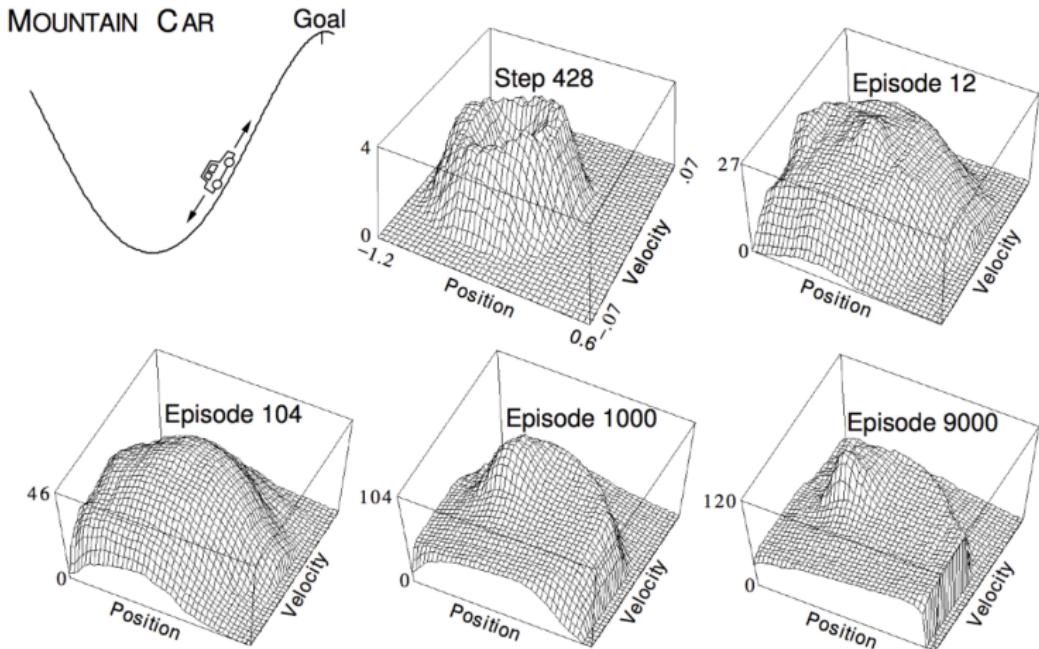
8: 更新策略 $\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$

9: $t \leftarrow t + 1$

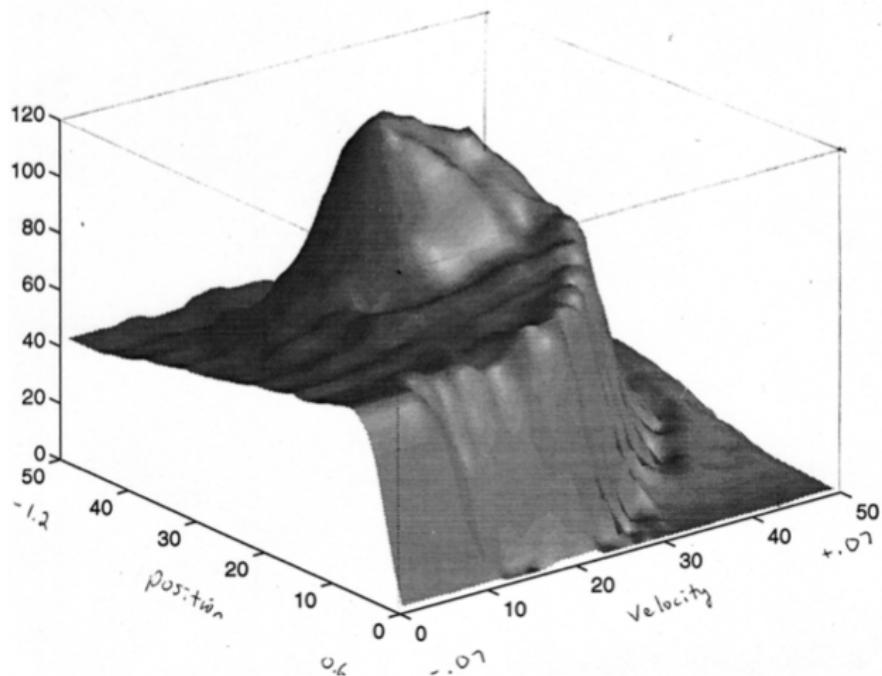
10: **end loop**

举例：小车爬山问题

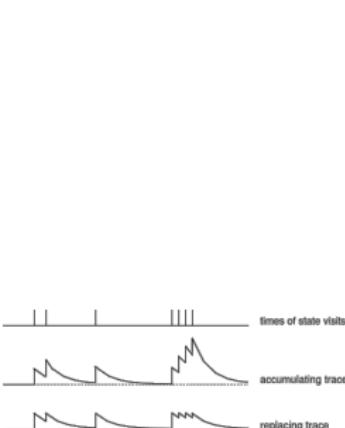
■ 线性 Sarsa+ 粗糙编码



■ 线性 Sarsa+RBF



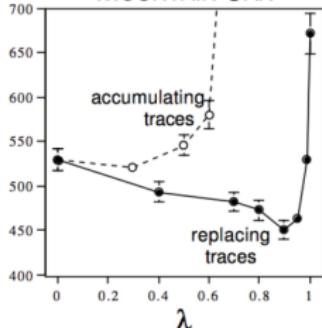
■ λ 对实验结果的影响



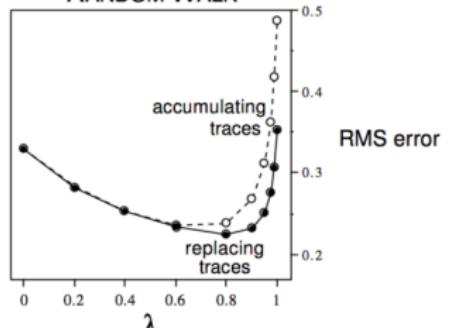
Steps per episode

Cost per episode

MOUNTAIN CAR



RANDOM WALK



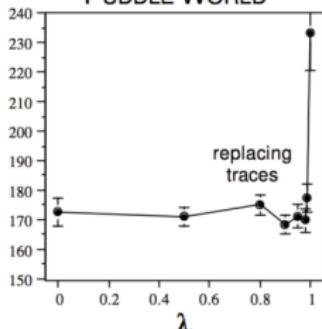
accumulating trace

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ \gamma \lambda e_{t-1}(s) + 1 & \text{if } s = s_t, \end{cases}$$

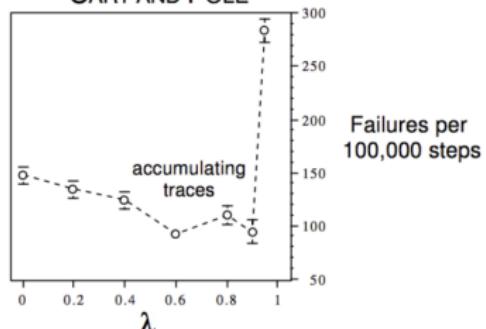
replacing trace

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ 1 & \text{if } s = s_t. \end{cases}$$

PUDDLE WORLD



CART AND POLE



■ 增加 λ 有助于降低收敛误差

■ 但 λ 过大会加大目标的方差, 不利于算法学习

- MC, TD(0), TD(λ) 都是在策略学习
- 离策略的 Q 学习同样可以和逼近器结合

- 1: 定义 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w})$, 初始化 \mathbf{w} ,
 $\pi_b = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$, $s_t = s_0$, $t = 0$
- 2: **loop**
- 3: 采样动作 $a_t \sim \pi(s_t)$ 并执行, 观测 (r_{t+1}, s_{t+1})
- 4: 根据 $(s_t, a_t, r_{t+1}, s_{t+1})$ 计算 TD 误差

$$\delta_t = r_{t+1} + \gamma \max_{a'} \hat{Q}(s_{t+1}, a', \mathbf{w}) - \hat{Q}(s_t, a_t, \mathbf{w})$$

- 5: 更新权重 $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta_t \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t, \mathbf{w})$
- 6: 更新策略 $\pi_b = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$
- 7: $t \leftarrow t + 1$
- 8: **end loop**

总结

函数逼近器

线性函数逼近

常见的特征表示方法

价值迭代 + 离散化方法

Fitted Q Iteration

策略迭代 + 最小二乘

预测学习 + 随机梯度下降法

控制学习 + 随机梯度下降法