强化学习

Reinforcement learning

第6节 价值函数近似

Value Function Approximation

张世周

Outlines

- 1.1 簡介
- 1.2 增量方法
- 1.3 批方法

大规模强化学习

强化学习可以用来解决大规模问题，比如：

西洋双陆棋有 10^{20}个状态空间

围棋有10^{170} 个状态空间

直升机特技表演是连续的环境

如何扩展上两节课中预测和控制的无模型方法到大规模问题中？

价值函数近似（Value Function Approximation）

到目前为止，我们使用的是查表（Table Lookup）的方式，这意味着每一个状态或者每一个状态-行为对对应表格中的一项。

对于大规模问题，这么做需要太多的内存来存储，而且有的时候针对逐个状态学习得到价值也是一个很慢的过程。

对于大规模问题，解决思路可以是这样的：

1. 通过函数近似来估计实际的价值函数：

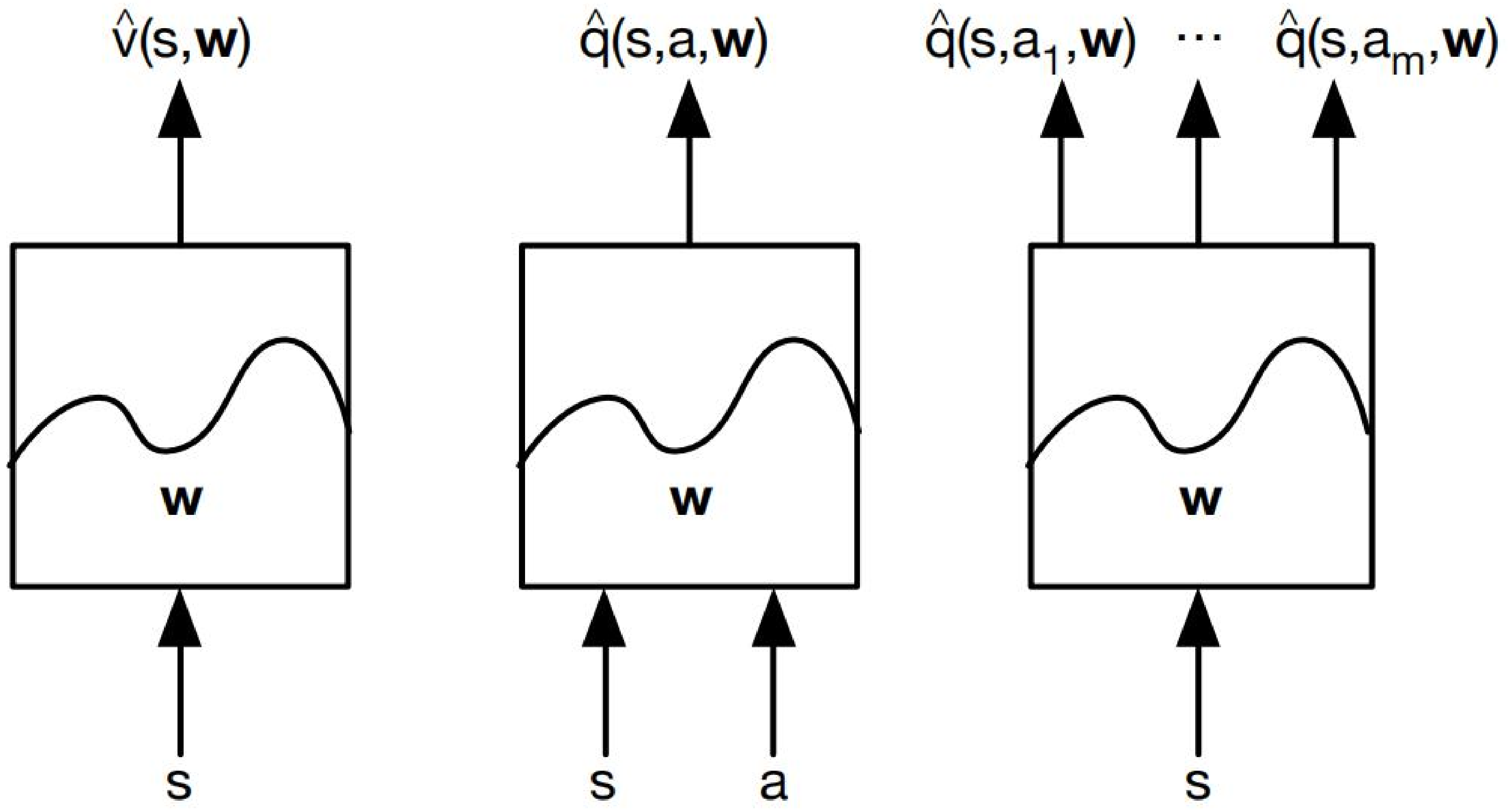
$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

2. 把从见过的状态中学到的函数，推广至那些未见过的状态中

3. 使用MC或TD学习方法来更新函数参数w。

近似价值函数的类型



$\hat{v}(s,\mathbf{w})$
$\hat{q}(s,a,\mathbf{w})$
$\hat{q}(s,a_1,\mathbf{w})$ ··· $\hat{q}(s,a_m,\mathbf{w})$
w
w
w
s
s a
s

有哪些近似函数

- 特征的线性组合
- 神经网络
- 决策树
- 近邻
- 傅里叶/小波基等等

如何选择近似函数

我们考虑可微函数逼近器

- 特征的线性组合（线性函数）
- 神经网络
- 决策树
- 近邻
- 傅里叶/小波基等等

此外，我们需要一个适用于非平稳、非独立均匀分布的数据的训练方法来优化近似函数。

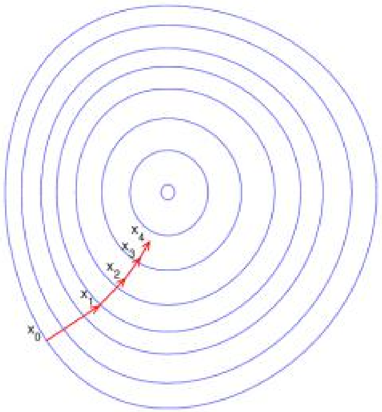
Outlines

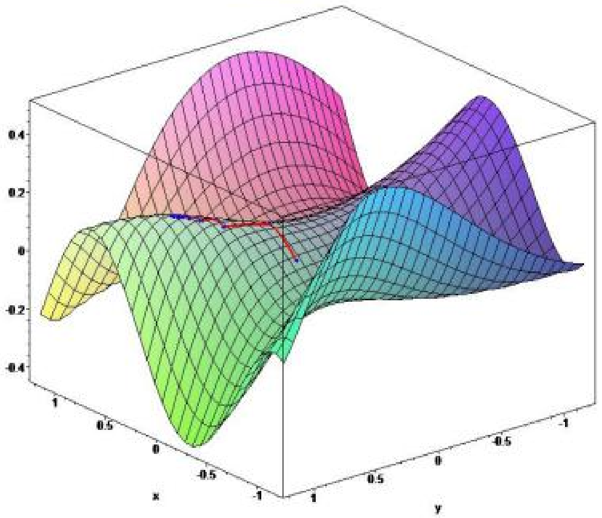
- 1.1 介绍
- 1.2 增量方法
- 1.3 批方法

梯度下降（Gradient Descent）

- 设J(w)是参数向量w的可微函数
- 定义J(w)的梯度为

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





- 求J(w)的局部极小值
- 沿-ve梯度方向调整w

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

- 其中α是步长参数

基于随机梯度下降的价值函数近似

目标：找到参数向量**w**，最小化近似函数 $\hat{v}(S,w)$ 与实际函数 $v_\pi(S)$ 的均方差：

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(v_\pi(S) - \hat{v}(S,\mathbf{w}))^2\right]$$

梯度下降能够找到局部最小值：

$$\begin{aligned}\Delta\mathbf{w} &= -\frac{1}{2}\alpha\nabla_\mathbf{w}J(\mathbf{w}) \\ &= \alpha\mathbb{E}_\pi\left[(v_\pi(S) - \hat{v}(S,\mathbf{w}))\nabla_\mathbf{w}\hat{v}(S,\mathbf{w})\right]\end{aligned}$$

随机梯度下降（SGD）对梯度进行采样：

$$\Delta\mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S,\mathbf{w}))\nabla_\mathbf{w}\hat{v}(S,\mathbf{w})$$

期望更新与全梯度更新相等

线性函数近似——特征向量

用特征向量表示状态，每一个状态是由以w表示的不同强度的特征来线性组合得到：

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

例如：

机器人与地标的距离

股市趋势

国际象棋中的棋子和棋子配置

线性函数近似

价值函数可以通过对特征的线性求和来近似表示：

$$\hat{v}(S,\mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$$

目标函数可以表示成参数w的二次函数：

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w}\right)^2\right]$$

使用随机梯度下降可以收敛至全局最优解

参数更新规则相对比较简单，即：参数更新量 = 步长 × 预测误差 × 特征值

$$\nabla_{\mathbf{w}} \hat{v}(S,\mathbf{w}) = \mathbf{x}(S)$$

$$\Delta\mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S,\mathbf{w}))\mathbf{x}(S)$$

“表格检索”特征

“表格检索”特征是一个特殊的线性价值函数近似方法：每一个状态看成一个特征，智能体具体处在某一个状态时，该状态特征取1，其余取0。

参数的数目就是状态数，也就是每一个状态特征有一个参数。

$$\mathbf{x}^{table}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix}$$

参数向量w给出每个单独状态的值

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

增量预测算法

- 上述算法都假定存在真实价值函数 $v_\pi(S)$，而强化学习没有监督数据，因此不能直接使用上述公式。强化学习里只有即时奖励，没有监督数据。我们要找到能替代 $v_\pi(S)$的目标值，以便来使用监督学习的算法学习到近似函数的参数。
- 对于MC，使用回报 G_t

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

- 对于TD(0)，使用TD目标值 $R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

- 对于TD(λ)，使用λ回报 G_t^λ

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

状态价值函数近似的MC方法

- 回报 G_t 是真实价值 $v_\pi(S_t)$的有噪声、无偏采样，可以把它看成是监督学习的标记数据带入监督学习算法进行学习，这样训练数据集可以是：

$$\langle S_1, G_1\rangle, \langle S_2, G_2\rangle, \ldots, \langle S_T, G_T\rangle$$

- 如果使用线性蒙特卡洛策略评估，那么每次参数的修正值则为：

$$\begin{aligned}\Delta \mathbf{w} &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)\end{aligned}$$

状态价值函数近似的TD方法

- TD目标值是真实价值 $v_\pi(S_t)$ 的有偏采样。此时的训练数据集是：

$$\langle S_1, R_2 + \gamma\hat{v}(S_2, \mathbf{w})\rangle, \langle S_2, R_3 + \gamma\hat{v}(S_3, \mathbf{w})\rangle, \ldots, \langle S_{T-1}, R_T\rangle$$

- 如果使用线性TD(0)学习，则有：

$$\begin{aligned}\Delta\mathbf{w} &= \alpha(R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S, \mathbf{w}) \\ &= \alpha\delta\mathbf{x}(S)\end{aligned}$$

- 线性TD(0)近似收敛至全局最优解。

状态价值函数近似的TD(λ)方法

- TD(λ)目标值是真实价值 $v_\pi(S_t)$的有偏采样。此时的训练数据集是：

$$\left\langle S_1, G_1^\lambda \right\rangle, \left\langle S_2, G_2^\lambda \right\rangle, \ldots, \left\langle S_{T-1}, G_{T-1}^\lambda \right\rangle$$

- 如果使用线性TD(λ)学习，从前向视角看，有：

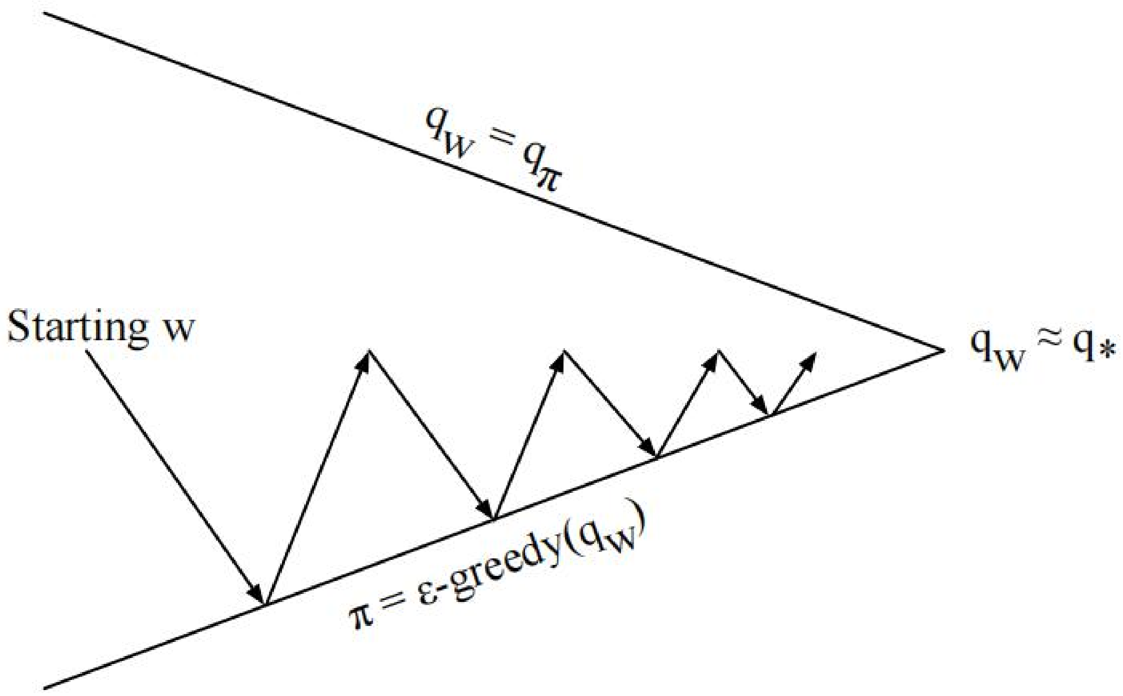
$$\begin{aligned}\Delta \mathbf{w} &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)\end{aligned}$$

- 如果使用线性TD(λ)学习，从后向视角看，有：

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \\ E_t &= \gamma \lambda E_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha \delta_t E_t\end{aligned}$$

- 对于一个完整的Episode，TD(λ)的前向视角和后向视角对于w的更新是等价的。

价值函数近似的控制方法



- 策略评估：近似策略评估，$\hat{q}(.,.,w) \approx q_\pi$
- 策略改善：使用ε-greedy执行。

动作价值函数近似

- 动作价值函数近似表示为：

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- 最小化近似动作价值值fn $\hat{q}(S, A, w)$和真实动作价值fn $q_\pi(S, A)$之间的均方误差

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2\right]$$

- 使用随机梯度下降来寻找局部最优解：

$$-\frac{1}{2}\nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$$

$$\Delta\mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$$

线性函数近似动作价值函数

- 状态-动作可以用特征向量表示：

$$\mathbf{x}(S,A) = \begin{pmatrix} \mathbf{x}_1(S,A) \\ \vdots \\ \mathbf{x}_n(S,A) \end{pmatrix}$$

- 如此，线性特征组合的状态行为价值近似函数可以表示为：

$$\hat{q}(S,A,\mathbf{w}) = \mathbf{x}(S,A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S,A)\mathbf{w}_j$$

- 随机梯度下降更新参数：

$$\nabla_{\mathbf{w}} \hat{q}(S,A,\mathbf{w}) = \mathbf{x}(S,A)$$

$$\Delta \mathbf{w} = \alpha(q_\pi(S,A) - \hat{q}(S,A,\mathbf{w}))\mathbf{x}(S,A)$$

增量控制算法

- 与预测算法类似，我们需要真实行为价值 $q_\pi(S,A)$ 的替代值。
- 对于MC算法，目标值就是回报：

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

- 对于TD(0)，目标值就是TD目标值：

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

- 对于前向视角TD(λ)，目标值是Q的λ回报：

$$\Delta \mathbf{w} = \alpha(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

- 对于后向视角TD(λ)，等价的参数更新是：

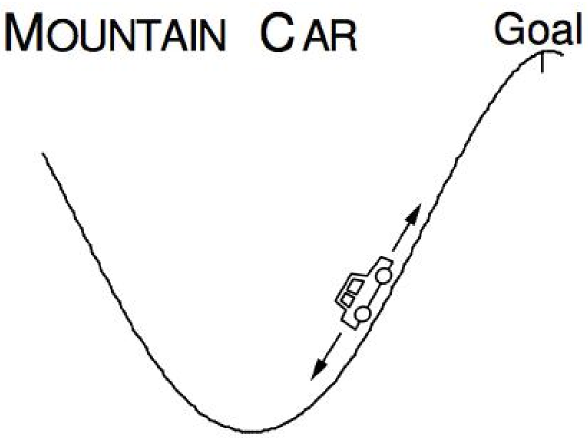
$$\delta_t = R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})$$

$$E_t = \gamma\lambda E_{t-1} + \nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

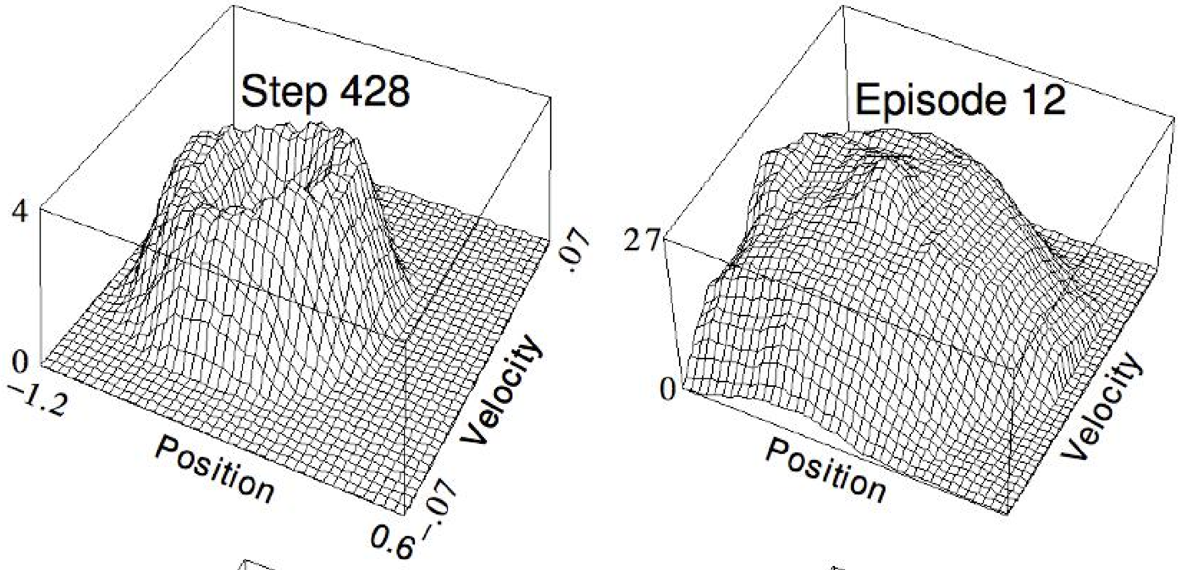
$$\Delta \mathbf{w} = \alpha\delta_t E_t$$

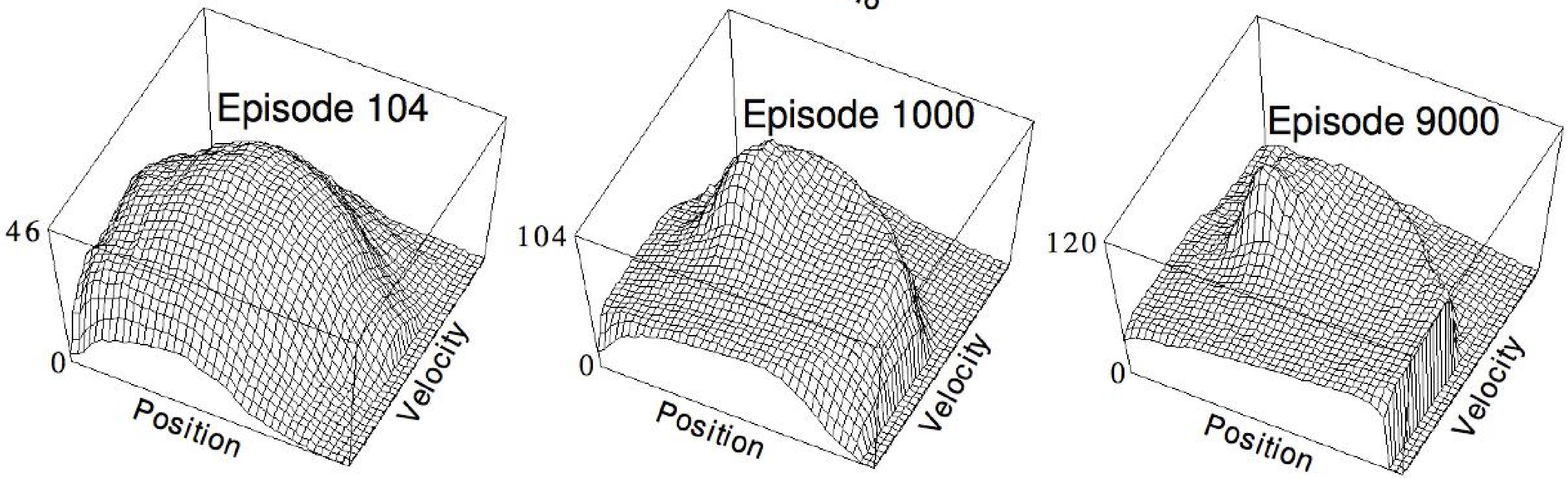
资格迹的扩展：
资格迹追踪了对最近状态评估值做出了或正或负贡献的权值向量的分量。当一个强化事件出现时，我们认为这些贡献"痕迹"展示了权值向量的对应分量有多少"资格"可以接受学习过程引起的变化。

小车爬山--使用粗编码的线性SARSA



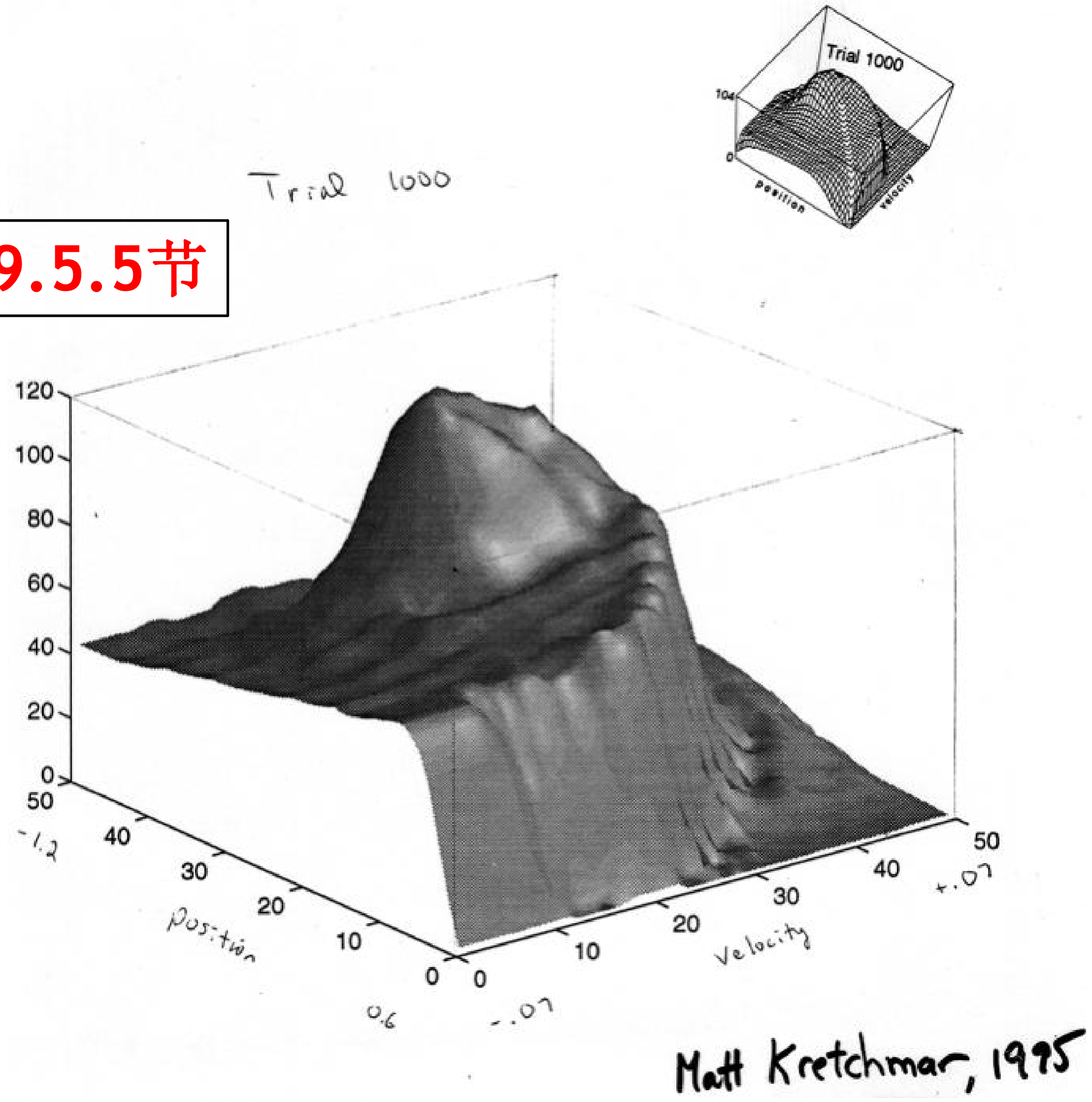
粗编码参考教材9.5.3-9.5.4节



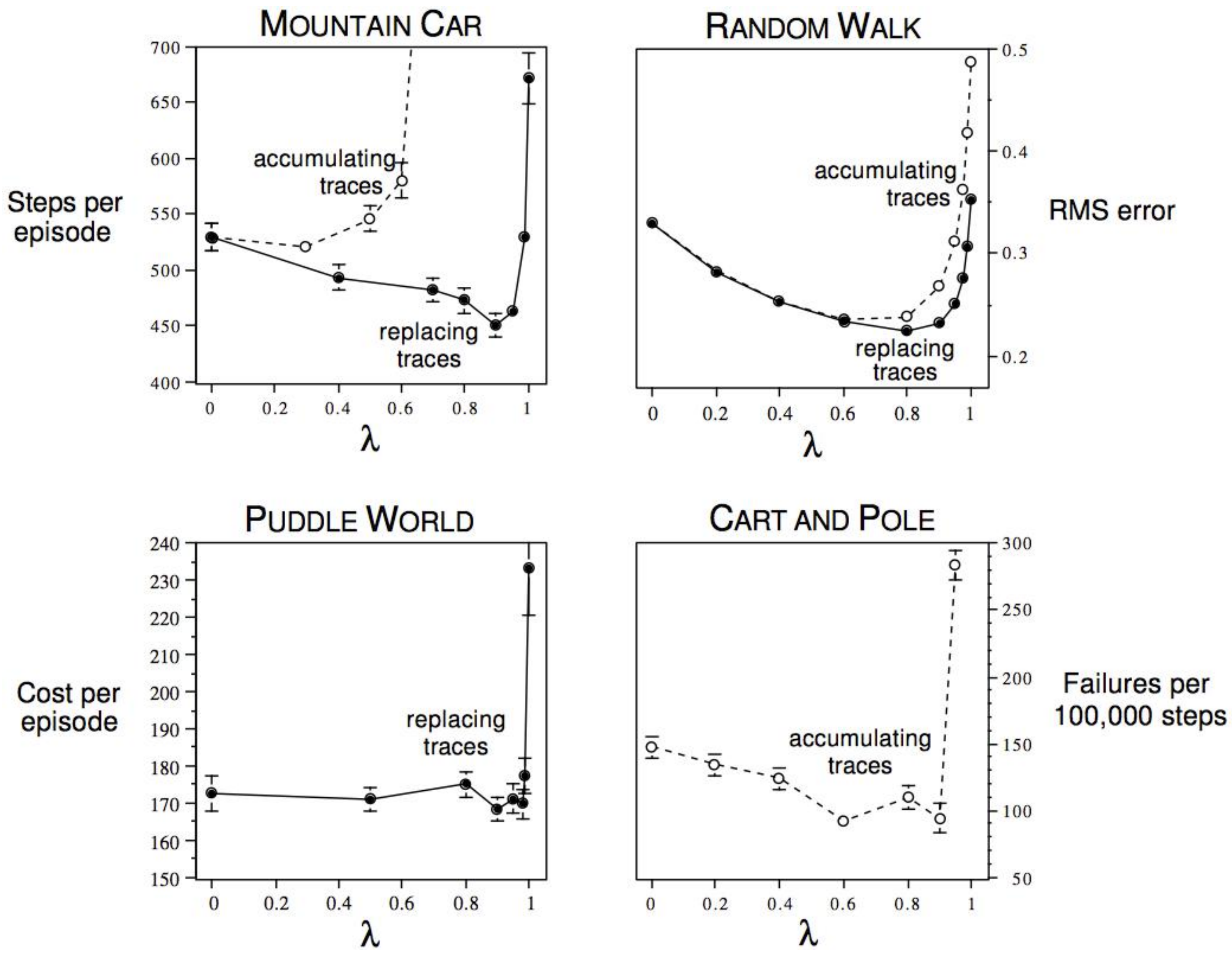


小车爬山--使用径向基函数的线性SARSA

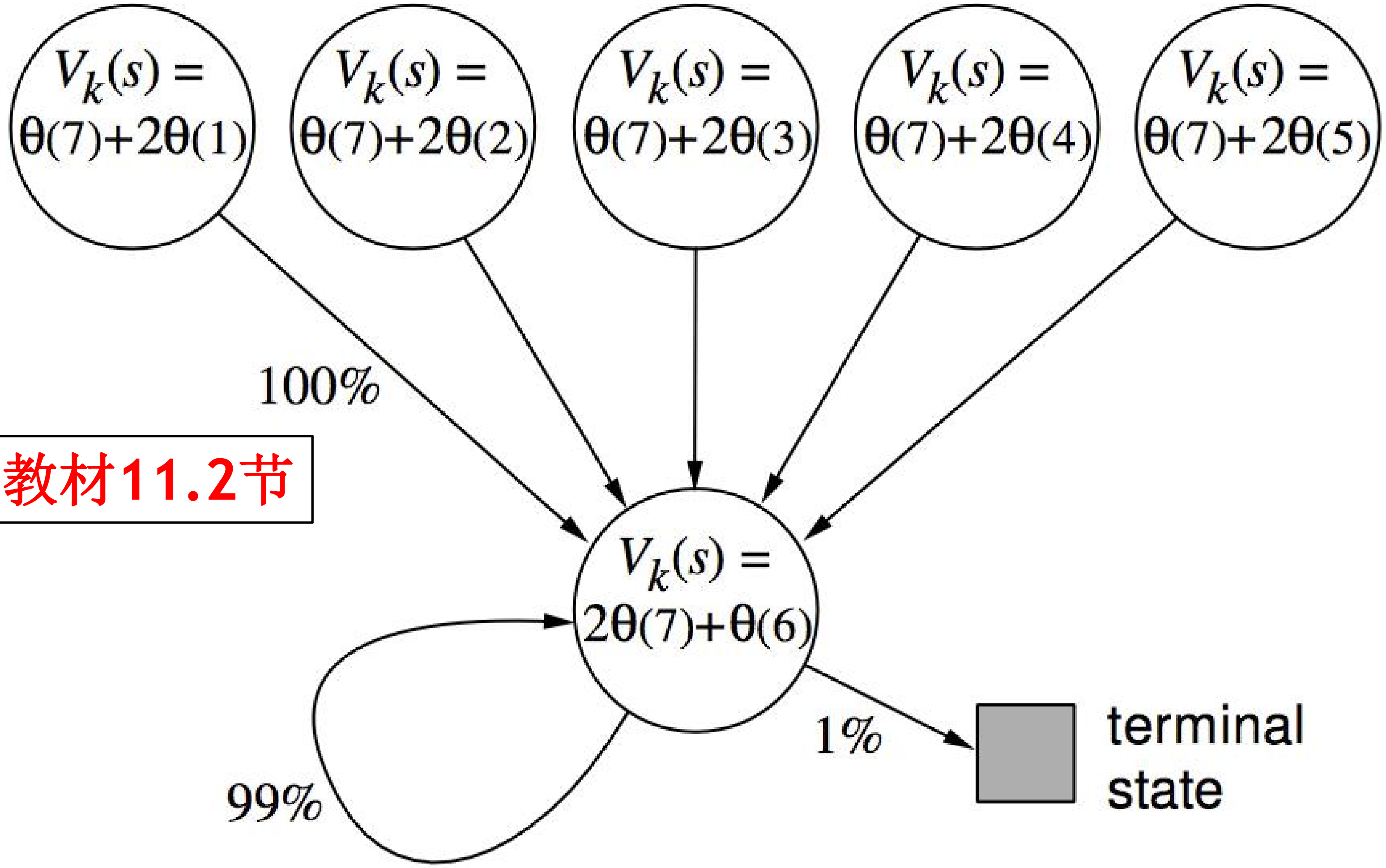
径向基函数参考教材9.5.5节



关于λ——需要Bootstrap吗？



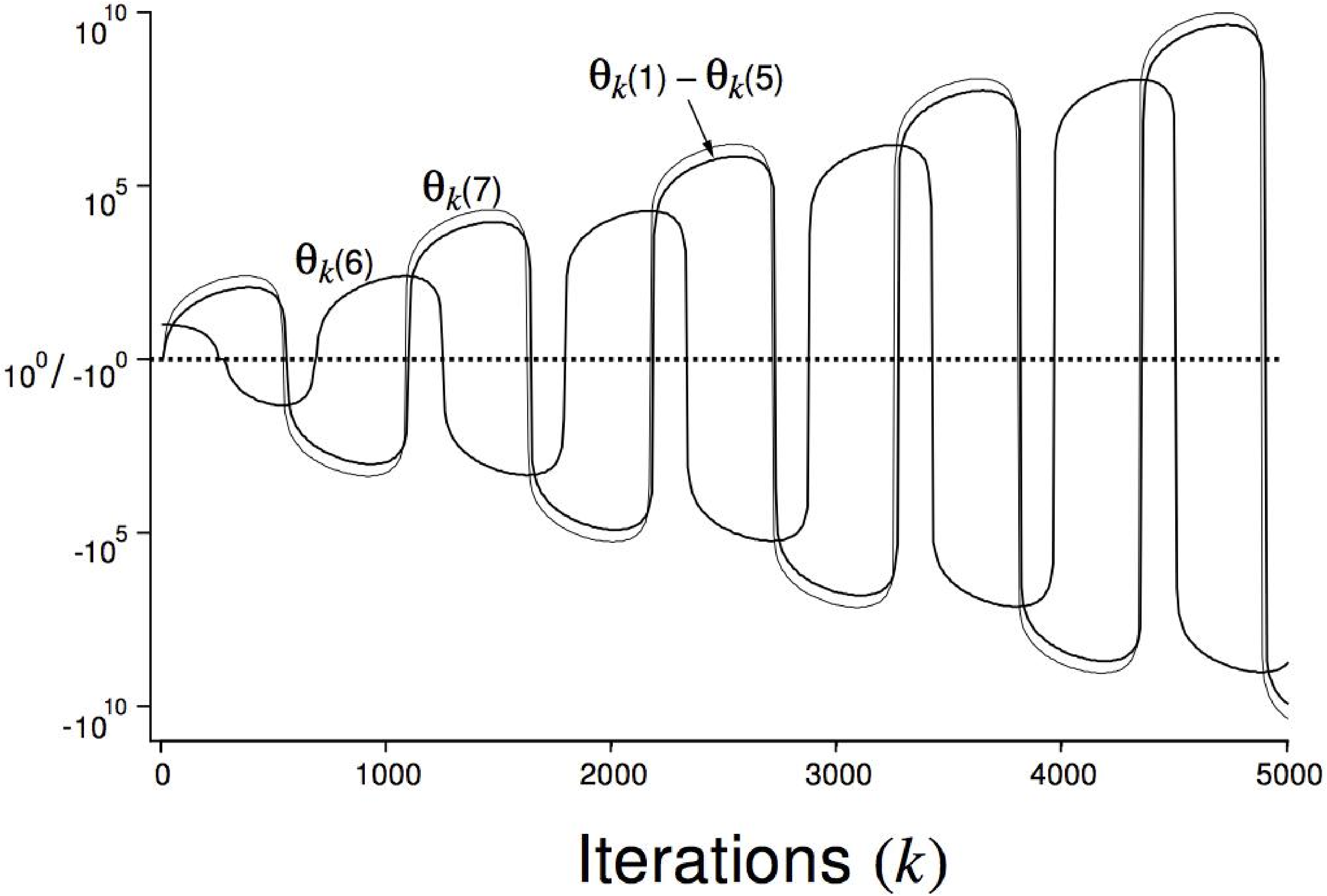
贝尔德反例---TD算法必然收敛吗？



Baird反例参考教材11.2节

贝尔德反例中的参数发散

Parameter values, $\theta_k(i)$ (log scale, broken at ±1)



预测算法的收敛性

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

梯度时间差分学习（Gradient TD）

TD不遵循任何目标函数的梯度，这就是TD在异策学习或使用非线性函数近似时会发散的原因，梯度TD遵循投影贝尔曼误差的真实梯度

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD	✓	✓	✗
	Gradient TD	✓	✓	✓
Off-Policy	MC	✓	✓	✓
	TD	✓	✗	✗
	Gradient TD	✓	✓	✓

Gradient TD参考教材11.7节

控制算法的收敛性

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

√=接近最优值函数的抖动

Outlines

- **1.1 介绍**
- **1.2 递增方法**
- **1.3 批方法**

批方法

- 前面所说的增量算法(Icremental Method)都是基于数据流的，经历一步，更新算法后，我们就不再使用这步的数据了，这种算法简单，但有时候不够高效。
- 与之相反，批方法(Batch Method)则是把一段时期内的数据集中起来，通过学习来使得参数能较好地拟合这段时期内所有的数据。这里的训练数据集“块”相当于智能体的一段经验。

最小二乘法预测

- 假设存在一个价值函数的近似：$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- 以及一段时期的，包含<状态、价值>对的经历D：

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- 最小二乘法算法要求找到参数w，使得下式值最小：

$$\begin{aligned} LS(\mathbf{w}) &= \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2 \\ &= \mathbb{E}_\mathcal{D} \left[(v^\pi - \hat{v}(s, \mathbf{w}))^2 \right] \end{aligned}$$

带经验回放的随机梯度下降

经验回放（Experience Replay）： 把一段时期内的经验重新过一遍，更新参数。

这种算法实现起来不是很难，只要重复以下过程：

1. 从经验中采样一个<s,v>：

$$\langle s, v^{\pi} \rangle \sim \mathcal{D}$$

2. 应用随机梯度下降来更新参数：

$$\Delta \mathbf{w} = \alpha(v^{\pi} - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

收敛至： 针对这段经验数据的最小二乘法的最优解：

$$\mathbf{w}^{\pi} = \underset{\mathbf{w}}{\operatorname{argmin}}\ LS(\mathbf{w})$$

DQN网络中的经验回放

DQN(Deep Q-Networks) 使用经验回放和固定的Q目标值

1. 依据Ɛ-greedy执行策略产生t时刻的动作a_t；

2. 将大量经历数据（例如百万级的）以 $(s_t, a_t, r_{t+1}, s_{t+1})$ 存储在回访内存D中；

3. 从D中随机抽取小批量（例如64个样本数据）数据 (s, a, r, s')；

4. 维护两个神经网络DQN1，DQN2,一个网络固定参数专门用来产生目标值，目标值相当于标签数据。另一个网络专门用来评估策略，更新参数。

5. 优化关于Q网络和Q目标值之间的最小均方误差：

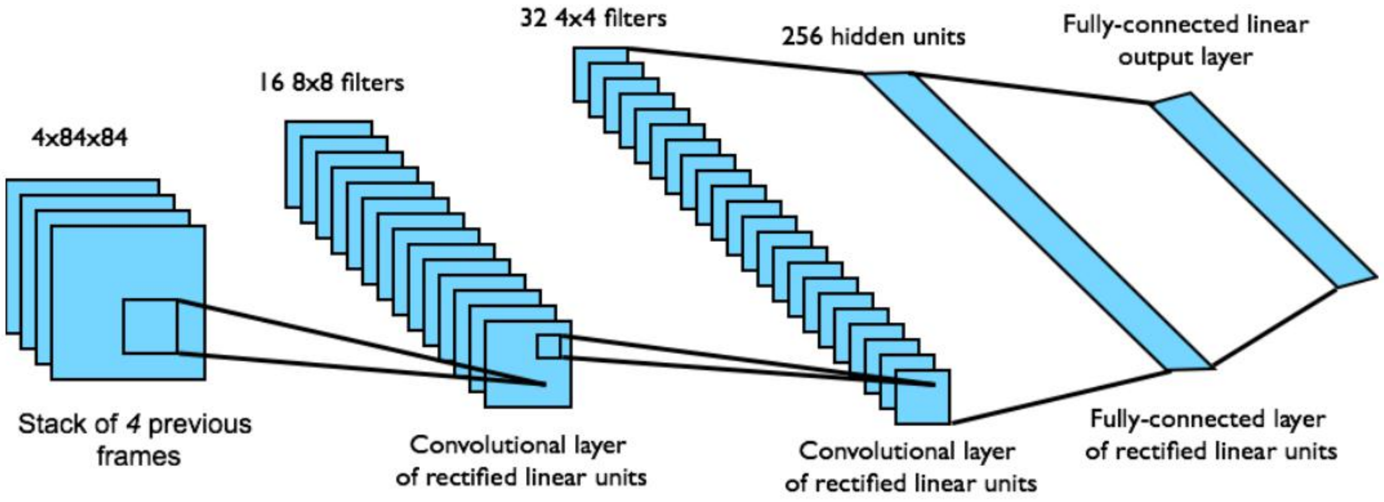
$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

式中：w_i^- 在mini-batch学习过程中是固定的，w_i 则是动态更新的参数。

6. 用随机梯度下降的方式更新参数

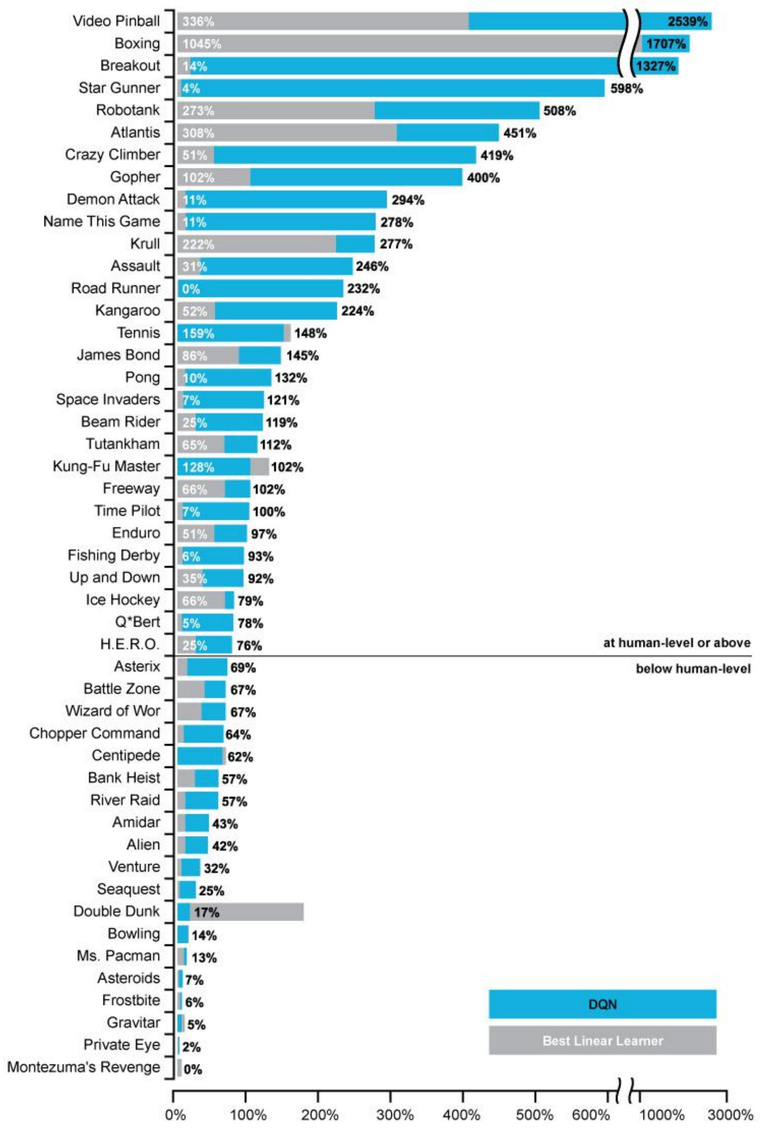
DQN在Atari中的应用

- 从像素s到值Q(s,a)的端到端学习
- 输入状态s是之前4帧的原始像素堆叠
- 输出为Q(s,a)对应的18个操纵杆/按钮位置
- 奖励信息是屏幕显示的分数变化



网络架构和超参数在所有游戏中都保持不变

DQN方法在Atari中的结果



DQN效果如何？

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

线性最小二乘法预测

通过比较发现使用批方法能够找到最小二乘法的解决方案，提高算法的稳定性，但是它需要多次迭代。我们可以设计一个价值函数的线性近似函数：

$$\hat{v}(s,w)=x(s)^T w$$

然后直接求解参数。

线性最小二乘法预测(2)

求解思路是逆向思维，假设已经找到这个参数，那么他应该满足最小LS(w)，通过把LS展开，可以直接得到w：

$$\mathbb{E}_{\mathcal{D}}\left[\Delta \mathbf{w}\right] = 0$$

$$\alpha \sum_{t=1}^{T} \mathbf{x}(s_t)(v_t^{\pi} - \mathbf{x}(s_t)^{\top}\mathbf{w}) = 0$$

$$\sum_{t=1}^{T} \mathbf{x}(s_t) v_t^{\pi} = \sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}(s_t)^{\top}\mathbf{w}$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}(s_t)^{\top}\right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t) v_t^{\pi}$$

这种方法直接求解的时间复杂度是 $O(N^3)$，使用Shermann-Morrison法求解复杂度是 $O(N^2)$

线性最小二乘法预测算法

因为我们不知道真实的价值 v_t^π ，所以事实上，我们的训练数据都是有噪声的，有偏的数据 v_t^π ：

LSMC Least Squares Monte-Carlo 使用回报 $v_t^\pi \approx G_t$
LSTD Least Squares Temporal-Difference 使用**TD**目标值 $v_t^\pi \approx R_{t+1} + \gamma\hat{v}(S_{t+1}, w)$
LSTD(λ) Least Squares TD(λ) 使用了**λ**-回报 $v_t^\pi \approx G_t^\lambda$

在每种情况下，直接求解MC/TD/TD（λ）的不动点（最优解）

线性最小二乘法预测算法（2）

LSMC

$$0 = \sum_{t=1}^{T} \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$$

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

LSTD

$$0 = \sum_{t=1}^{T} \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(S_t)(\mathbf{x}(S_t) - \gamma\mathbf{x}(S_{t+1}))^\top\right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t)R_{t+1}$$

LSTD(λ)

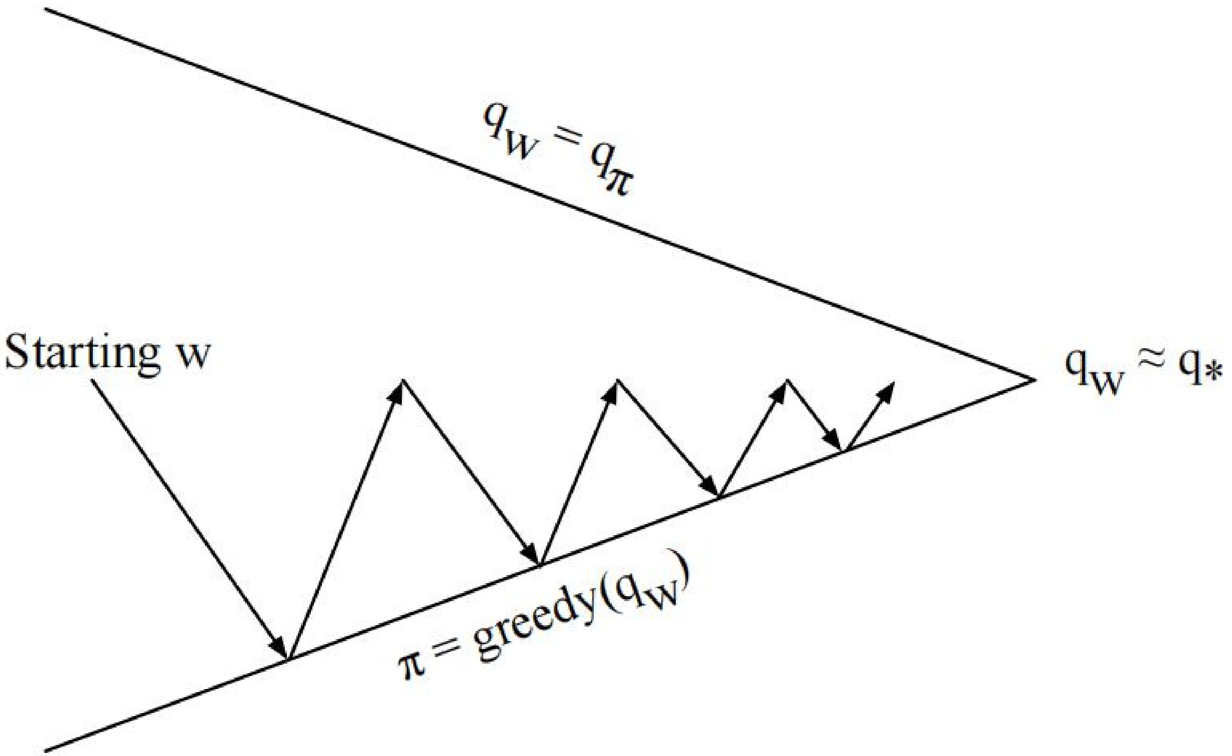
$$0 = \sum_{t=1}^{T} \alpha\delta_t E_t$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} E_t(\mathbf{x}(S_t) - \gamma\mathbf{x}(S_{t+1}))^\top\right)^{-1} \sum_{t=1}^{T} E_t R_{t+1}$$

线性最小二乘法预测算法的收敛性

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✓	✗
	LSTD	✓	✓	-
Off-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✗	✗
	LSTD	✓	✓	-

最小二乘法策略迭代



策略评估使用：最小二乘法Q学习
策略改善使用：Greedy 策略改进。

最小二乘法动作价值函数近似

- 动作价值函数近似 $q_\pi(s,a)$
- 使用特征的线性组合 $x(s,a)$

$$\hat{q}(s,a,w) = x(s,a)^T w \approx q_\pi(s,a)$$

- 最小化 $\hat{q}(s,a,w)$和 $q_\pi(s,a)$之间的最小二乘误差，根据使用策略**π**产生的经验由<(state,action),value>对组成

$$\mathcal{D} = \{\langle (s_1, a_1), v_1^\pi \rangle, \langle (s_2, a_2), v_2^\pi \rangle, \ldots, \langle (s_T, a_T), v_T^\pi \rangle\}$$

最小二乘法控制

（1）对于策略评估，我们想尽可能的使用所有的经验；（2）对于控制，我们想提升策略；（3）经验来自于多个策略；（4）评估 $q_\pi(s,a)$必须使用异策学习。

我们使用与Q-学习类似的想法：

- 使用由旧策略生成的经验 $S_t, A_t, R_{t+1}, S_{t+1} \sim \pi_{old}$
- 考虑替代后继动作 $A' = \pi_{new}(S_{t+1})$
- 利用 $R_{t+1} + \gamma\hat{q}(S_{t+1}, A', \mathbf{w}))$ 的值更新 $\hat{q}(S_t, A_t, \mathbf{w})$

最小二乘Q学习

下面是线性Q学习的更新式

$$\delta = R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})$$
$$\Delta \mathbf{w} = \alpha \delta \mathbf{x}(S_t, A_t)$$

LSTDQ算法：求解　总更新=零

$$0 = \sum_{t=1}^{T} \alpha (R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\mathbf{x}(S_t, A_t)$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(S_t, A_t)(\mathbf{x}(S_t, A_t) - \gamma \mathbf{x}(S_{t+1}, \pi(S_{t+1})))^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t, A_t) R_{t+1}$$

最小二乘策略迭代算法

以下伪代码使用LSTDQ进行策略评估

它反复使用不同的策略重新评估经验D

function LSPI-TD($\mathcal{D}, \pi_0$)

  $\pi' \leftarrow \pi_0$

  repeat

    $\pi \leftarrow \pi'$

    $Q \leftarrow$ LSTDQ($\pi, \mathcal{D}$)

    for all $s \in \mathcal{S}$ **do**

      $\pi'(s) \leftarrow \operatorname*{argmax}_{a \in \mathcal{A}} Q(s, a)$

    end for

  until $(\pi \approx \pi')$

  return π

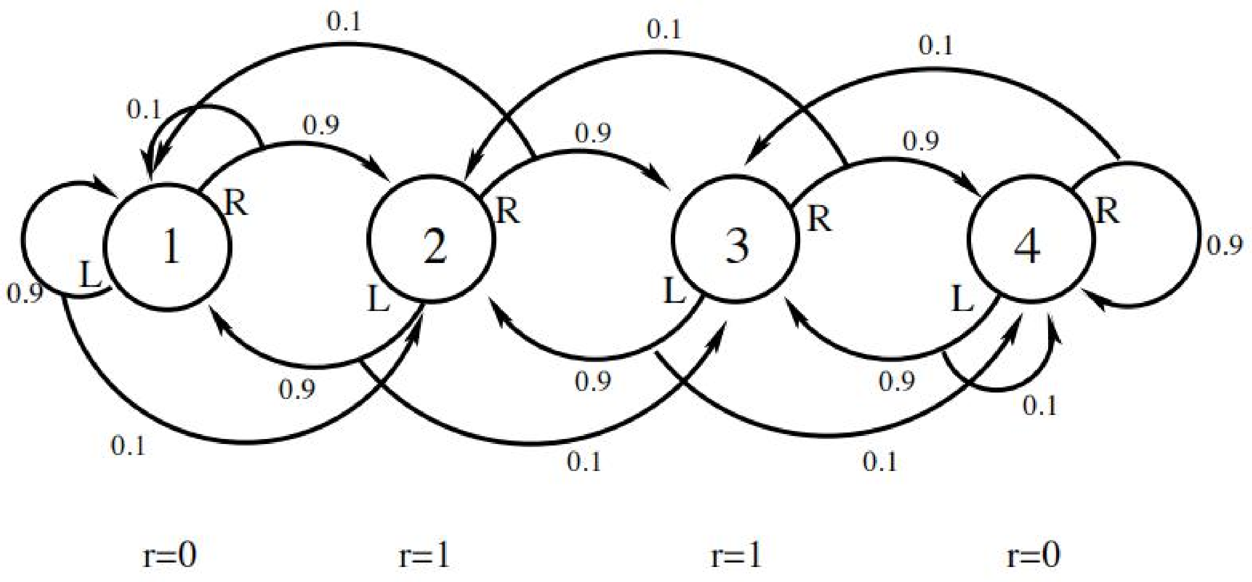
end function

控制算法的收敛性

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
LSPI	✓	(✓)	-

（√）=在接近最优值函数附近位置抖动

链式行走示例



这个问题有50个状态

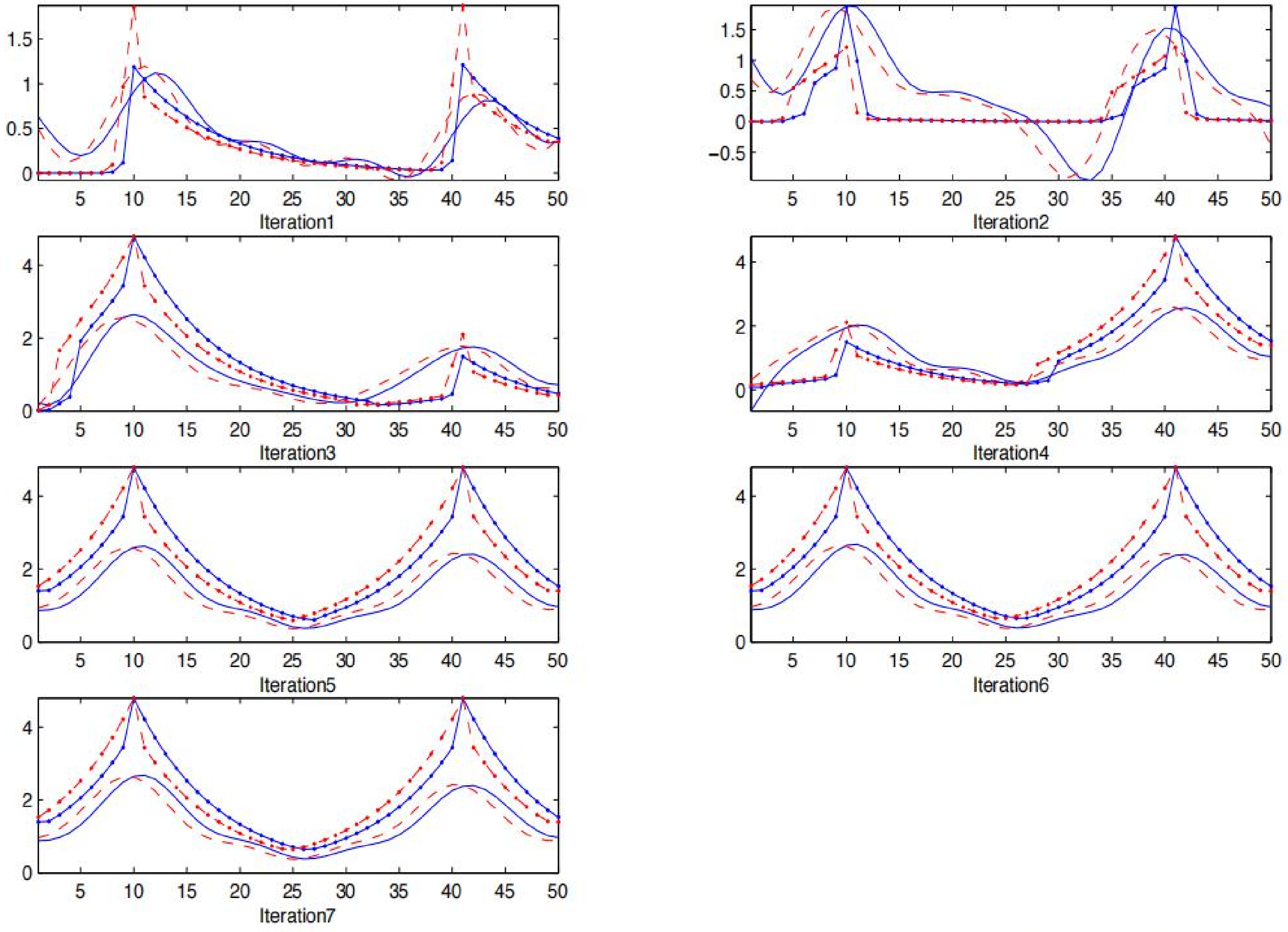
在第10和第41状态时获得+1的收益，其余状态为0

最优策略：R（1-9），L（10-25），R（26-41），L（42,50）

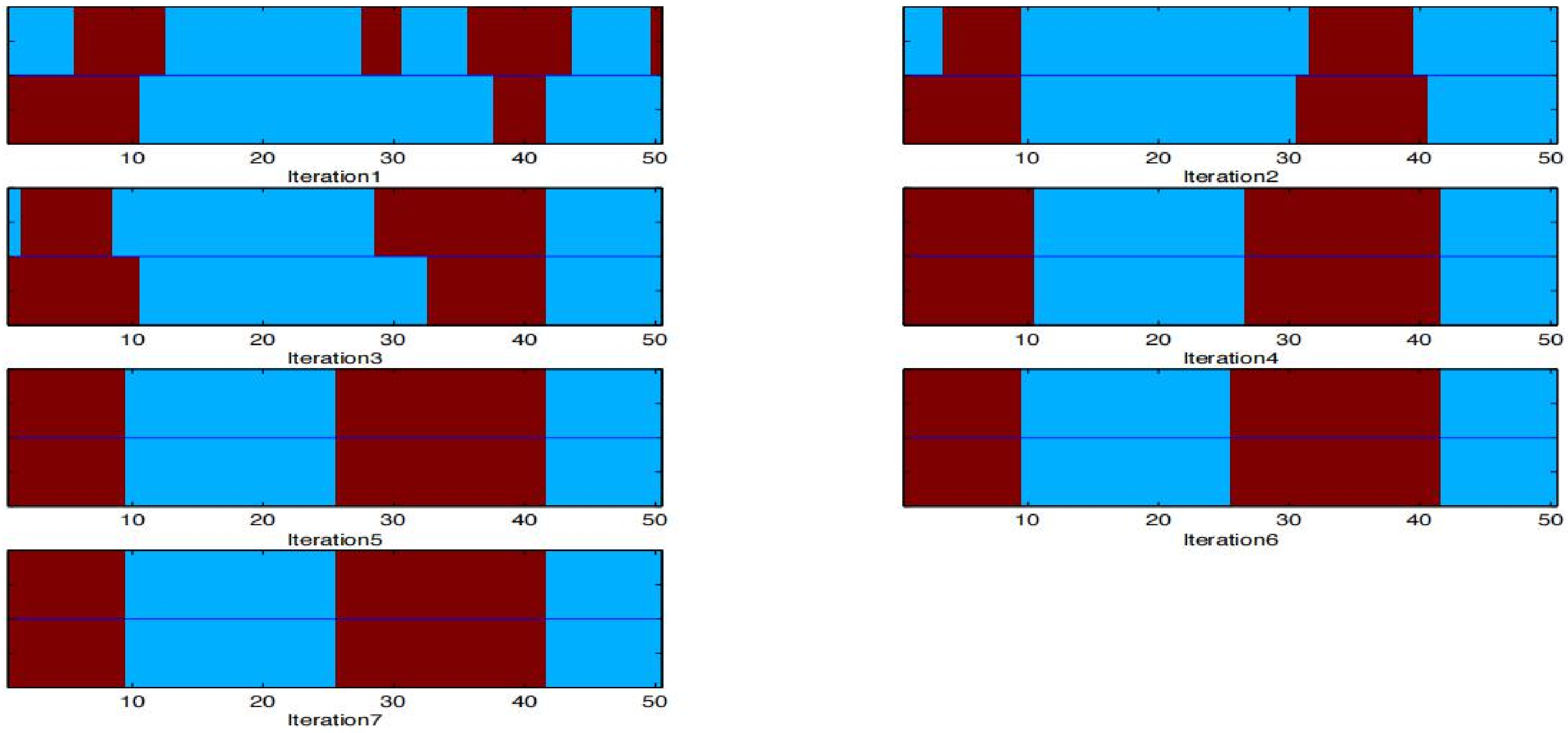
特征：每个动作10个等距高斯（σ=4）

经验：随机策略走10000步

LSPI应用在链式行走：动作价值函数



LSPI应用在链式行走：策略



The End