值函数估计（Value Function Approximation）

吉建民

USTC
jianmin@ustc.edu.cn

2023 年 10 月 31 日

Used Materials

Disclaimer: 本课件大量采用了 Rich Sutton's RL class, David Silver's Deep RL tutorial 和其他网络课程课件，也采用了 GitHub 中开源代码，以及部分网络博客内容

Table of Contents

大规模状态的强化学习

- 如果把之前的强化学习算法用于如下问题时，它面临的状态空间？
  - atari 游戏: 若一张图片是 $200x200$, 状态空间为 $(255^3)^{200x200}$
  - 围棋: 10^{170} 个状态
  - 无人车: 连续的状态空间
- Model-free 方法 MC 和 TD 在 prediction 和 control 时，需要存储相应值函数

值函数估计定义

- 以前，值函数被保存在一张表中
  - $V(s)$ for each state s
  - $Q(s, a)$ for each state-action pair s, a
- 这种方法的优点是，简单而且可以保存所有可能的值
- 但如果 states 十分大的时候，会存在如下问题
  - 要很大的空间去保存值函数
  - 对每个状态都进行学习的话，需要的时间长

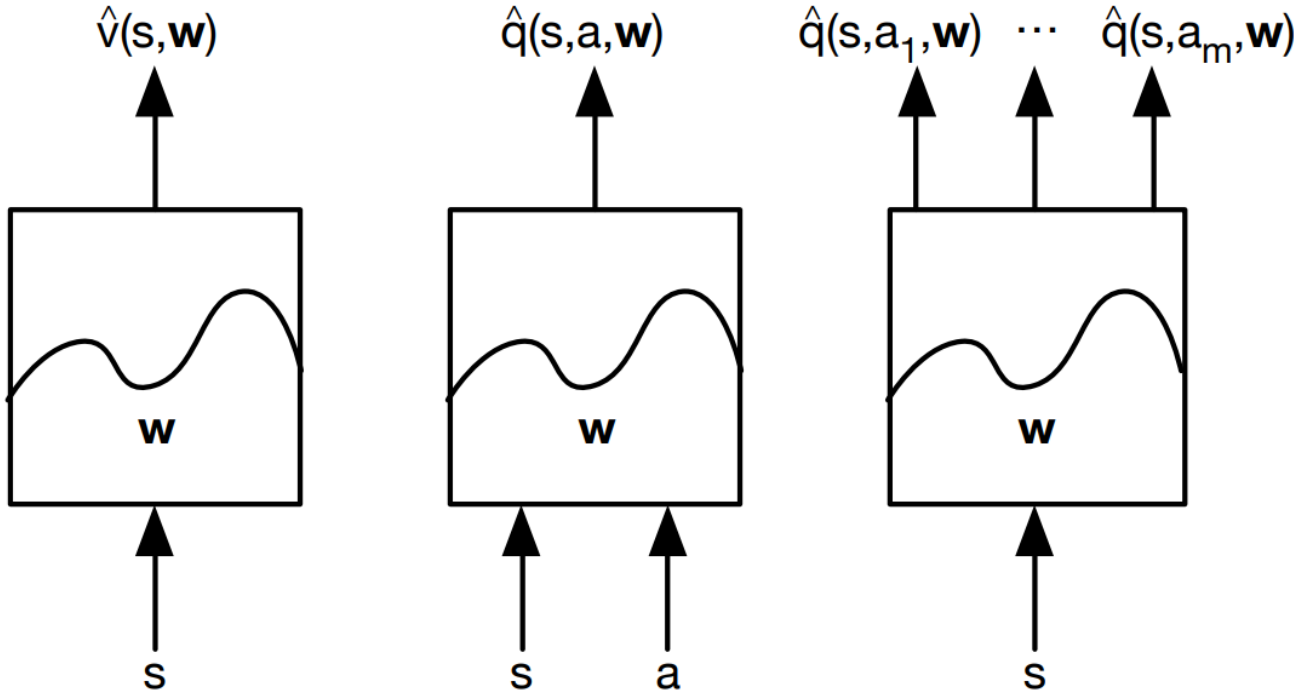
值函数估计定义

- 在实际应用中，对于状态和行为空间都比较大的情况下，精确获得各种 $V(s)$ 和 $Q(s,a)$ 几乎是不可能的。这时候需要找到近似的函数，具体可以使用线性组合、神经网络以及其他方法来近似价值函数
- 对大规模状态的 MDP 的处理方法：
  - 可以利用监督学习里的数据拟合方法去估计值函数

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$$
$$\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

  - 数据拟合方法具有泛化能力，一些没出现过的状态也有可能被正确估计
  - 可以利用后面的 MC learning 或 TD learning 更新参数 $\boldsymbol{w}$
- 通过函数近似，可以用少量的参数 $\boldsymbol{w}$ 来拟合实际的各种价值函数

值函数种类



- 针对状态本身，输出这个状态的近似价值
- 针对状态行为对，输出状态行为对的近似价值
- 针对状态本身，输出一个向量，向量中的每一个元素是该状态下采取一种可能行为的价值

哪种方法去做值函数估计？

- 在机器学习和模式识别中，有许多方法可用于函数近似:
  - 线性模型
  - 神经网络
  - 决策树
  - 最近邻分类
  - ……
- 所有和机器学习相关的一些算法都可以应用到强化学习中来，其中线性模型和神经网络在强化学习里应用得比较广泛，主要是考虑这两类方法是一个针对状态可导的近似函数
- 强化学习应用的场景其数据通常是非静态（non-stationary）、非独立均匀分布（non-iid）的，因为一个状态数据是可能是持续流入的，而且下一个状态通常与前一个状态是高度相关的。因此，我们需要一个适用于非静态、非独立均匀分布的数据的训练方法来得到近似函数
- 下面将分别从递增方法和批方法两个角度来讲解价值函数的近似方法，其主要思想都是梯度下降，与机器学习中的随机梯度下降和批梯度下降相对应

Table of Contents

梯度下降

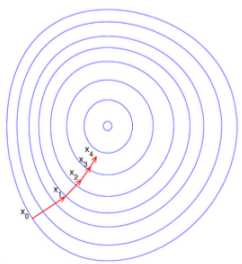
- $J(w)$ 是关于参数向量 w 的可导函数，那么 $J(w)$ 的梯度如下:

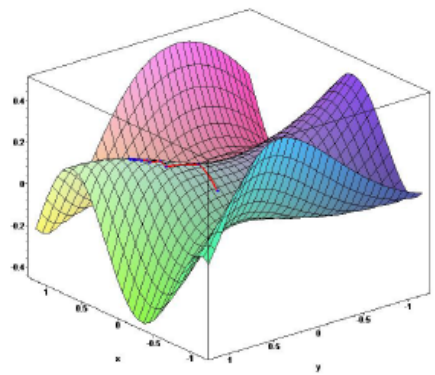
$$\nabla J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}$$

- 梯度代表函数变化最大的方向，故为了找到 $J(w)$ 的局部最小点，可以沿着梯度相反的方向调整 w

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

- 其中 α 是步长参数，机器学习里称为学习速率参数





梯度下降法（Gradient Descent）

- 给定可导函数 $F(x)$，其极值点 $\nabla F(x) = 0$（还有鞍点 Saddle Point）
- $F(x)$ 在 a 点沿着梯度相反的方向 $-\nabla F(a)$ 下降最快，以最快速度下降到局部最小值
- 对于足够小的 $\gamma > 0$，如果 $b = a - \gamma \nabla F(a)$，则 $F(a) \geq F(b)$
- 因此，可以从某个初始值 x_0 出发，找到函数 F 的局部最小值

$$x_{k+1} = x_k - \gamma_k \nabla F(x_k)$$

可以得到

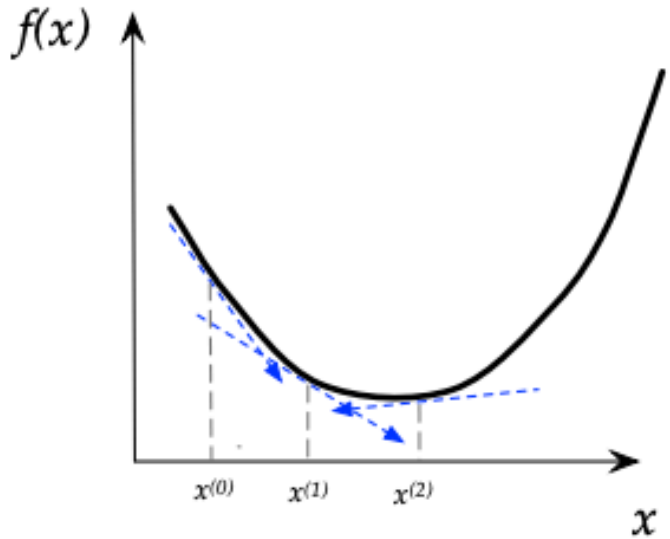
$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \cdots,$$

最终收敛到一个局部最小值

梯度下降法（Gradient Descent）

Algorithm 1 Gradient Descent

1: Guess $\mathbf{x}^{(0)}$, set $k \leftarrow 0$
2: **while** $||\nabla f(\mathbf{x}^{(k)})|| \geq \epsilon$ **do**
3: $\quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - t_k \nabla f(\mathbf{x}^{(k)})$
4: $\quad k \leftarrow k + 1$
5: **end while**
6: **return** $\mathbf{x}^{(k)}$



随机梯度下降法（Stochastic Gradient Descent）

- 随机梯度下降（SGD）不仅可节省计算，还可避免陷入局部极值，收敛到全局极值（当目标函数为凸或伪凸函数）
  - 对于 $F(x) = \frac{1}{n}\sum_{i=1}^{n} F_i(x)$ 形式的目标函数
    - 相当于机器学习中总共有 $1, \ldots, n$ 的数据，F_i 为第 i 次的观察
  - 按梯度下降定义，迭代公式为：

$$x_{k+1} = x_k - \gamma \nabla F(x_k) = x_k - \gamma \sum_{i=1}^{n} \frac{\nabla F_i(x_k)}{n}$$

  - 随机梯度下降迭代公式为：

$$x_{k+1} = x_k - \gamma \nabla F_i(x_k)$$

  对随机选择的 $1 \leq i \leq n$

利用随机梯度下降的值函数估计

- VFA(value function approximation) 目标: 寻找一个参数向量 w, 使得估计值 $\hat{v}(s, \boldsymbol{w})$ 和真实值 $v_\pi(s)$ 之间的均方误差最小

$$J(\boldsymbol{w}) = E_\pi[(v_\pi(s) - \hat{v}(s, \boldsymbol{w}))^2]$$

- 梯度下降能够找到局部最小值:

$$\begin{aligned}\Delta \boldsymbol{w} &= -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \\ &= \alpha E_\pi[(v_\pi(s) - \hat{v}(s, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{v}(s, \boldsymbol{w})]\end{aligned}$$

- 随机梯度下降（stochastic gradient descend)

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

  - 随机梯度下降对梯度进行更新，来近似差的期望，每一步，参数朝着实际的价值函数进行一定程度地逼近

特征向量

- 把状态 s 表示为一个特征向量

$$x(s) = \begin{pmatrix} x_1(s) \\ x_2(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

- 例如:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

线性函数估计

- 用线性特征表示值函数

$$\hat{v}(s, \boldsymbol{w}) = x(s)^\top \boldsymbol{w} = \sum_{j=1}^{n} x_j(s) w_j$$

- 那么目标函数 $J(w)$ 如下

$$J(\boldsymbol{w}) = E_\pi[(v_\pi(s) - x(s)^\top \boldsymbol{w})^2]$$

- 对于线性模型，使用随机梯度下降可以收敛至全局最优解，参数更新规则如下:

$$\begin{aligned} \nabla_{\boldsymbol{w}} \hat{v}(s, \boldsymbol{w}) &= x(s) \\ \Delta \boldsymbol{w} &= \alpha(v_\pi(s) - \hat{v}(s, \boldsymbol{w}))x(s) \\ &= \alpha(v_\pi(s) - x(s)^\top \boldsymbol{w})x(s) \end{aligned}$$

即：参数更新量 = 步长 × 预测误差 × 特征值

查表特征（Table Lookup Features）

- “查表”方法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，个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
- 查表特征

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

- Parameter vector $\boldsymbol{w}$ gives value of each individual state

$$\hat{v}(s, \boldsymbol{w}) = \begin{pmatrix} \mathbf{1}(s = s_1) \\ \vdots \\ \mathbf{1}(s = s_n) \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$$

Table of Contents

What's the true value function $v_\pi(s)$

- 监督学习中，真实的样本值 $v_\pi(s)$ 可由人工标注
- 但在强化学习中，没有人工标注数据，只有 reward
- 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（target），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, $v_\pi(s)$ target 为 G_t

$$\Delta w = \alpha(G_t - \hat{v}(s, \boldsymbol{w}))\nabla_w \hat{v}(s, \boldsymbol{w})$$

  - 对于 TD(0), $v_\pi(s)$ target 为 $R_{t+1} + \gamma\hat{v}(s_{t+1}, \boldsymbol{w})$

$$\Delta w = \alpha(R_{t+1} + \gamma\hat{v}(s_{t+1}, \boldsymbol{w}) - \hat{v}(s, \boldsymbol{w}))\nabla_w \hat{v}(s, \boldsymbol{w})$$

  - 对于 TD(λ), $v_\pi(s)$ target 为 G_t^λ

$$\Delta w = \alpha(G_t^\lambda - \hat{v}(s, \boldsymbol{w}))\nabla_w \hat{v}(s, \boldsymbol{w})$$

VFA for prediction : MC

- 收获 G_t 是真实值 $v_\pi(s_t)$ 的有噪声、无偏（高方差）采样，可以把它看成是监督学习的标签数据带入机器学习算法进行学习，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估计（Linear Monte-Carlo policy evaluation），那么每次参数的修正值则为：

$$\begin{aligned} \Delta w &= \alpha(G_t - \hat{v}(s, \boldsymbol{w}))\nabla_w \hat{v}(s, \boldsymbol{w}) \\ &= \alpha(G_t - \hat{v}(s, \boldsymbol{w}))x(s_t) \end{aligned}$$

- MC 可以收敛到一个局部最优点（local optimum）

Monte Carlo with VFA

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^n \to \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta}$ as appropriate (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat forever:
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
    For $t = 0, 1, \ldots, T-1$:
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \big[G_t - \hat{v}(S_t, \boldsymbol{\theta}) \big] \nabla \hat{v}(S_t, \boldsymbol{\theta})$

VFA for prediction : TD

- TD-target $R_{t+1} + \gamma \hat{v}(s_{t+1}, \boldsymbol{w})$ 是真实值 $v_\pi(s_t)$ 的有偏估计
- 然而，仍可把 $R_{t+1} + \gamma \hat{v}(s_{t+1}, \boldsymbol{w})$ 当做监督学习中的训练样本值

$$\langle S_1, R_2 + \gamma \hat{v}(s_2, \boldsymbol{w}) \rangle, \langle S_2, R_3 + \gamma \hat{v}(s_3, \boldsymbol{w}) \rangle, \ldots, \langle S_{T-1}, R_T \rangle$$

- 如果使用线性 TD(0) 学习（linear TD(0)），则有：

$$\begin{aligned} \Delta \boldsymbol{w} &= \alpha(R_{t+1} + \gamma \hat{v}(s_{t+1}, \boldsymbol{w}) - \hat{v}(s, \boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{v}(s, \boldsymbol{w}) \\ &= \alpha(R_{t+1} + \gamma \hat{v}(s_{t+1}, \boldsymbol{w}) - \hat{v}(s, \boldsymbol{w})) x(s_t) \end{aligned}$$

- 线性 TD(0) 近似收敛至全局最优解（global optimum）

TD Learning with VFA

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^n \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\boldsymbol{\theta}$ arbitrarily (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{v}(S',\boldsymbol{\theta}) - \hat{v}(S,\boldsymbol{\theta})\big]\nabla\hat{v}(S,\boldsymbol{\theta})$
    $S \leftarrow S'$
  until S' is terminal

We ignore the dependence of the target on θ
We call it semi-gradient methods

VFA for prediction : TD(λ)

- λ-回报 G_t^λ 是真实值 $v_\pi(s_t)$ 的有偏估计
- 然而，仍可把 G_t^λ 当做监督学习中的训练样本值

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

- 如果使用前向线性 TD(λ) 学习（Forward view linear TD(λ)）有：

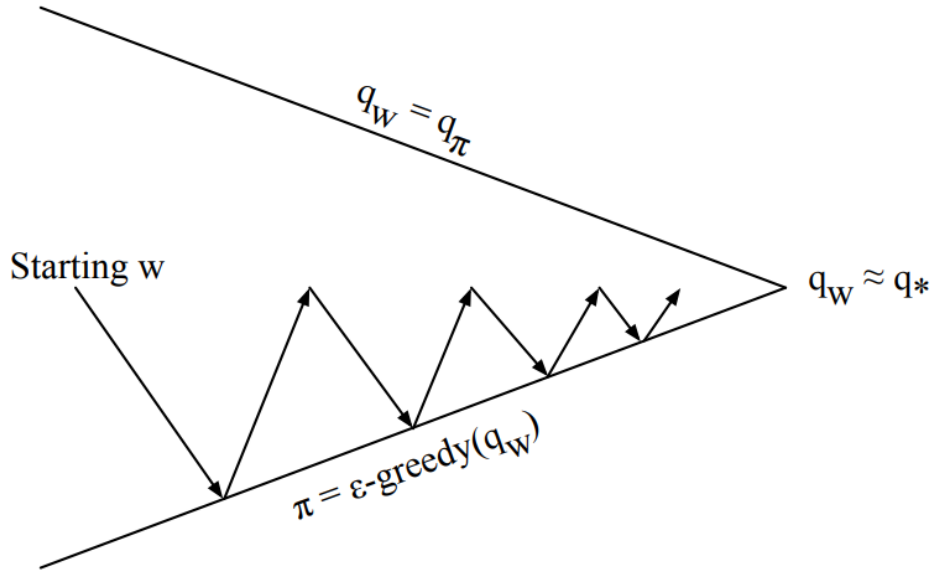
$$\begin{aligned}\Delta \boldsymbol{w} &= \alpha(G_t^\lambda - \hat{v}(s, \boldsymbol{w}))\nabla_w \hat{v}(s, \boldsymbol{w}) \\ &= \alpha(G_t^\lambda - \hat{v}(s, \boldsymbol{w}))x(s_t)\end{aligned}$$

- 如果使用后向线性 TD(λ) 学习（Backward view linear TD(λ)），有：

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

- 前向线性 TD(λ) 和后向线性 TD(λ) 是一样的
- 线性 TD(λ) 近似收敛至全局最优解。

VFA for control



- 策略估计 (Policy evaluation)： 用估计方法去做计算 q 值，$\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$
- 策略提升 (Policy improvement)： 仍用之前的 ϵ-greedy 做策略提升

行动值函数（Action-Value Function）估计

- 估计行动值函数

$$\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

- 最小化估计的 Action-Value 函数 $\hat{q}(s, a, \boldsymbol{w})$ 和真实的 Action-Value 函数 $q_\pi(s, a)$ 之间的均方误差

$$J(w) = E_\pi[(q_\pi(s, a) - \hat{q}(s, a, \boldsymbol{w}))^2]$$

- 用随机梯度下降去寻找最优值

$$-\frac{1}{2}\nabla_w J(w) = (q_\pi(s, a) - \hat{q}(s, a, \boldsymbol{w}))\nabla_w \hat{q}(s, a, \boldsymbol{w})$$

$$\Delta w = \alpha(q_\pi(s, a) - \hat{q}(s, a, \boldsymbol{w}))\nabla_w \hat{q}(s, a, \boldsymbol{w})$$

线性行动值函数估计

- 把状态和动作表示为一个特征向量

$$x(s,a) = \begin{pmatrix} x_1(s,a) \\ x_2(s,a) \\ \vdots \\ x_n(s,a) \end{pmatrix}$$

- 那么行动值函数可以表示为该特征向量和 $\boldsymbol{w}$ 的乘积

$$\hat{q}(s,a,\boldsymbol{w}) = x(s,a)^\top \boldsymbol{w} = \sum_{j=1}^{n} x_j(s,a) w_j$$

- 随机梯度下降更新参数：

$$\nabla_{\boldsymbol{w}} \hat{q}(s,a,\boldsymbol{w}) = x(s,a)$$

$$\Delta \boldsymbol{w} = \alpha(q_\pi(s,a) - \hat{q}(s,a,\boldsymbol{w}))x(s,a)$$

增量控制算法

- 跟之前的 prediction 部分一样，我们需要用 target 来代替 $q_\pi(s, a)$
- 对于 MC, target 为 G_t

$$\Delta \boldsymbol{w} = \alpha(G_t - \hat{q}(s_t, a_t, \boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{q}(s_t, a_t, \boldsymbol{w})$$

- 对于 TD(0), target 为 TD-target $R_{t+1} + \gamma\hat{q}(s_{t+1}, a_{t+1}\boldsymbol{w})$

$$\Delta \boldsymbol{w} = \alpha(R_{t+1} + \gamma\hat{q}(s_{t+1}, a_{t+1}, \boldsymbol{w}) - \hat{q}(s_t, a_t, \boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{q}(s_t, a_t, \boldsymbol{w})$$

- 对于前向 TD(λ), target 为 action-value $\lambda-$return

$$\Delta \boldsymbol{w} = \alpha(q_t^\lambda - \hat{q}(s_t, a_t, \boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{q}(s_t, a_t, \boldsymbol{w})$$

- 对于后向 TD(λ)，对应的参数更新是：

$$\delta_t = R_{t+1} + \gamma\hat{q}(s_{t+1}, a_{t+1}, \boldsymbol{w}) - \hat{q}(s_t, a_t, \boldsymbol{w})$$
$$E_t = \gamma\lambda E_{t-1} + \nabla_w\hat{q}(s_t, a_t, \boldsymbol{w})$$
$$\Delta \boldsymbol{w} = \alpha\delta_t E_t$$

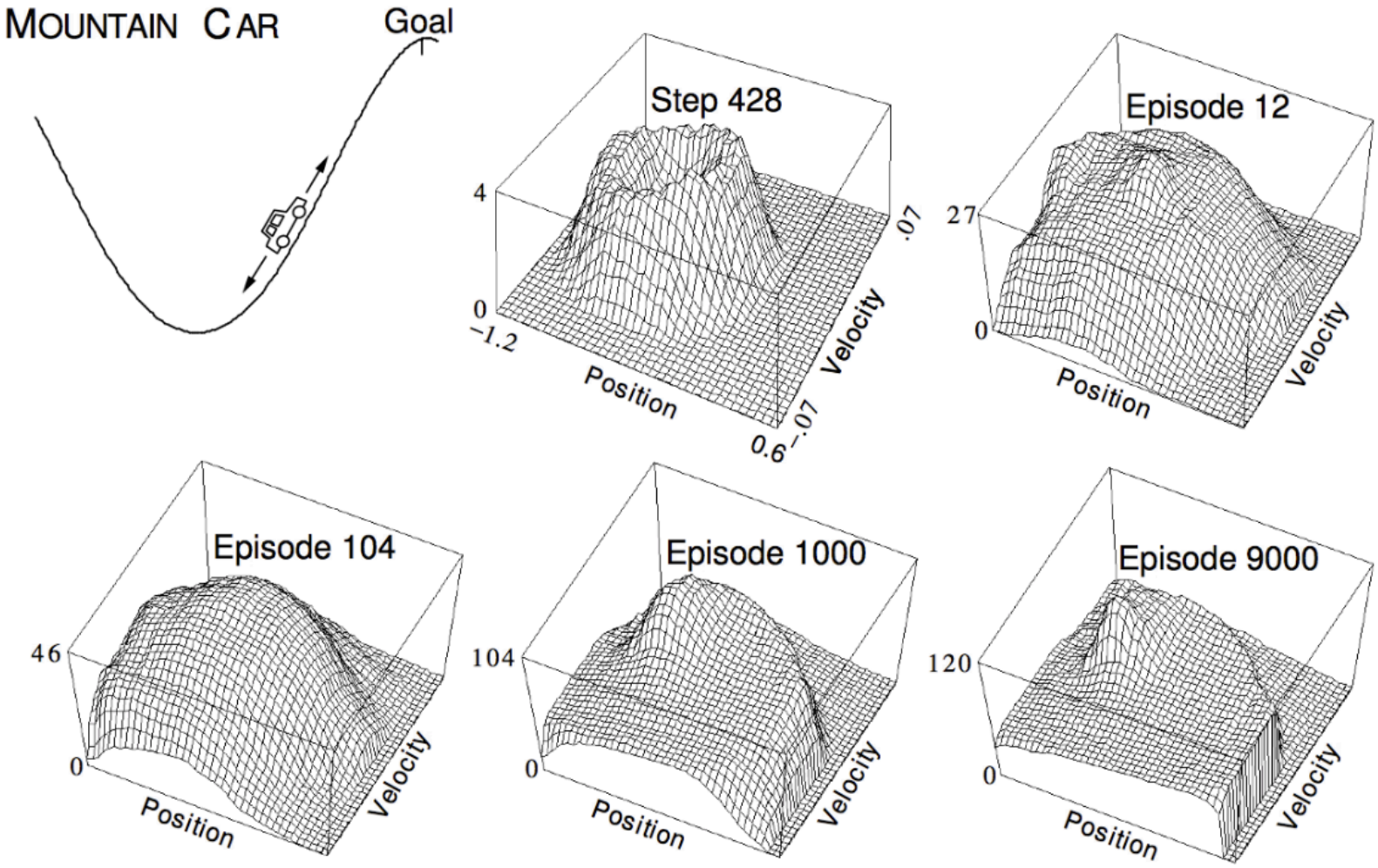
Incremental Control Algorithms

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

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \to \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta} \in \mathbb{R}^n$ arbitrarily (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat (for each episode):
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        If S' is terminal:
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
            Go to next episode
        Choose A' as a function of $\hat{q}(S', \cdot, \boldsymbol{\theta})$ (e.g., ε-greedy)
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{q}(S', A', \boldsymbol{\theta}) - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
        $S \leftarrow S'$
        $A \leftarrow A'$

示例—小车爬山

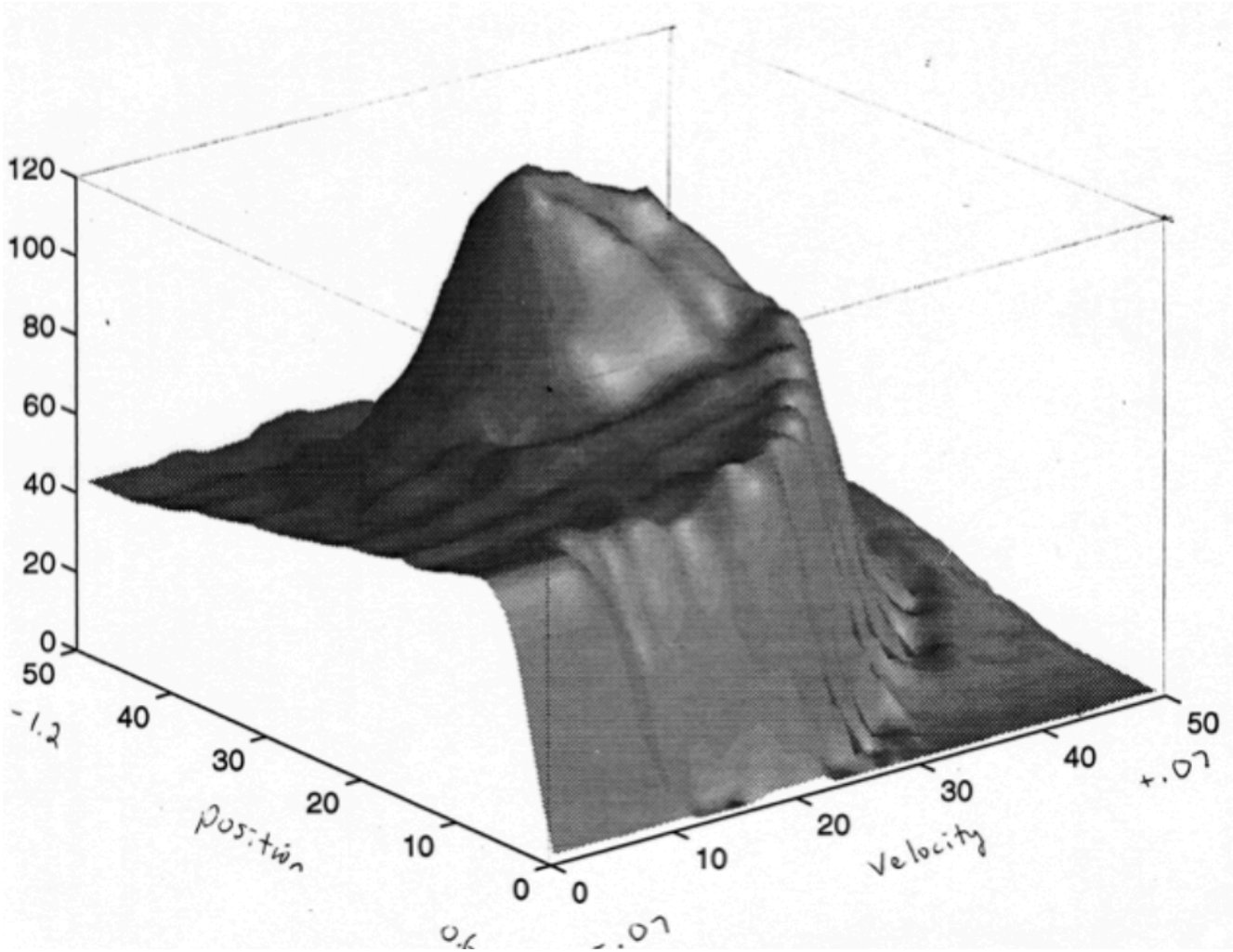

MOUNTAIN CAR
Goal
Step 428
4
0
−1.2
Position
0.6
−.07
Velocity
.07
Episode 12
27
0
Position
Velocity
Episode 104
46
0
Position
Velocity
Episode 1000
104
0
Position
Velocity
Episode 9000
120
0
Position
Velocity

示例—小车爬山

- 小车爬山是一个经典的强化学习示例。环境如图左上角所示，小车被困于山谷，单靠小车自身的动力是不足以在谷底由静止一次性冲上右侧目标位置的，比较现实的策略是，当小车加速上升到一定位置时，让小车回落，同时反向加速，使其加速冲向谷底，借助势能向动能的转化冲上目标位置。现在问题是在模型位置的情况下，如何用强化学习的方法找到小车冲上目标位置的最优策略？
- 状态空间是小车的位置和速度，其它几张三维图展示的是经过不同步数（上中图）以及不同 Episode（其余几张三维图）的学习，小车位于某个位置同时具有某个速度的状态价值。在这个例子中，行为空间是离散的，只能选则正向的最大油门和反向的最大油门两个离散值

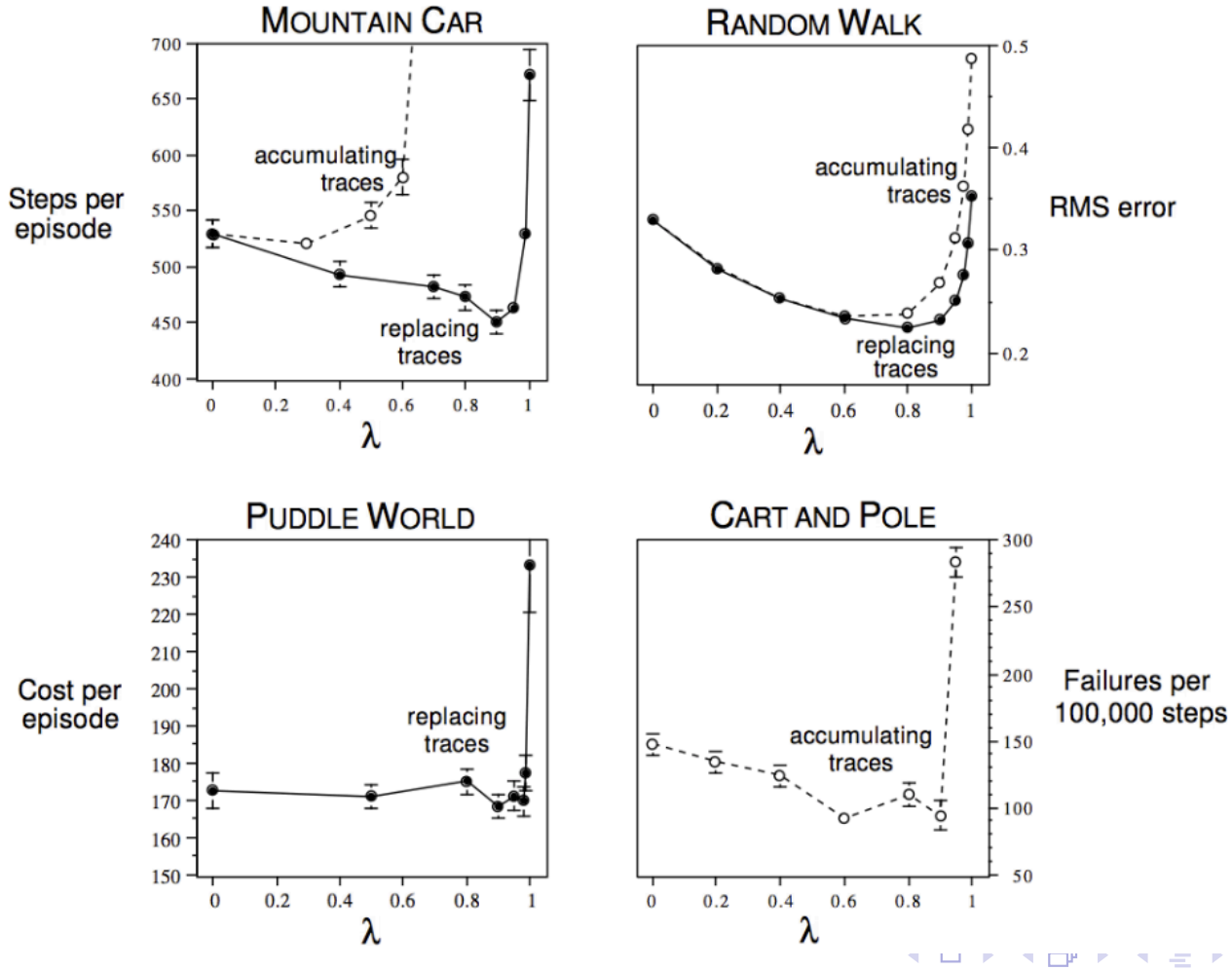
Linear Sarsa in Mountain Car

小车使用 SARSA 学习到了接近最优策略的价值函数，如下图：



关于 λ—需要 Bootstrap 吗？

下图显示了几种不同的任务，使用不同 λ 进行的强化学习算法分析结果。总的来说 λ=1 的时候通常算法表现是很差的，TD(0) 是比 MC 好得多的方法，这说明了 Bootstrap 的重要性；不同的任务对应的最优 λ 值是不太容易确定的。



Prediction 算法的收敛性

MC 使用的是实际价值的有噪声无偏估计，虽然很多时候表现很差，但总能收敛至局部或全局最优解。TD 性能通常更加优秀，是否意味着 TD 也是一直收敛的呢？答案是否定的。

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

从表中可以看出，没有函数近似时，各种算法都收敛；线性函数近似时现时策略学习可以收敛，但离线策略时仅有 MC 收敛；非线性函数近似时无论采用现时策略还是离线策略只有 MC 收敛。而 MC 算法在实际中是很少使用的。这给强化学习的实际应用带来的挑战。好在我们有一些改善 TD 算法的办法。

Gradient Temporal-Difference Learning

TD 算法在更新参数时不遵循任何目标函数的梯度是导致它在离线策略或使用非线性近似函数可能会发散的原因，我们可以通过修改 TD 算法使得它遵循 Projected Bellman Error 的梯度进而收敛。

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

Gradient Temporal-Difference Learning

Objective function:

- Close to the true values, Mean-Square Error:

$$MSE(\theta) = \sum_s d_s(V_\theta(s) - V(s))^2 = ||V_\theta - V||_D^2$$

- Close to satisfying the Bellman equation, Mean-Square Bellman Error:

$$MSBE(\theta) = ||V_\theta - TV_\theta||^2$$

  - 其中 T 通过 Bellman operator 定义

$$V = r + \gamma PV = TV$$

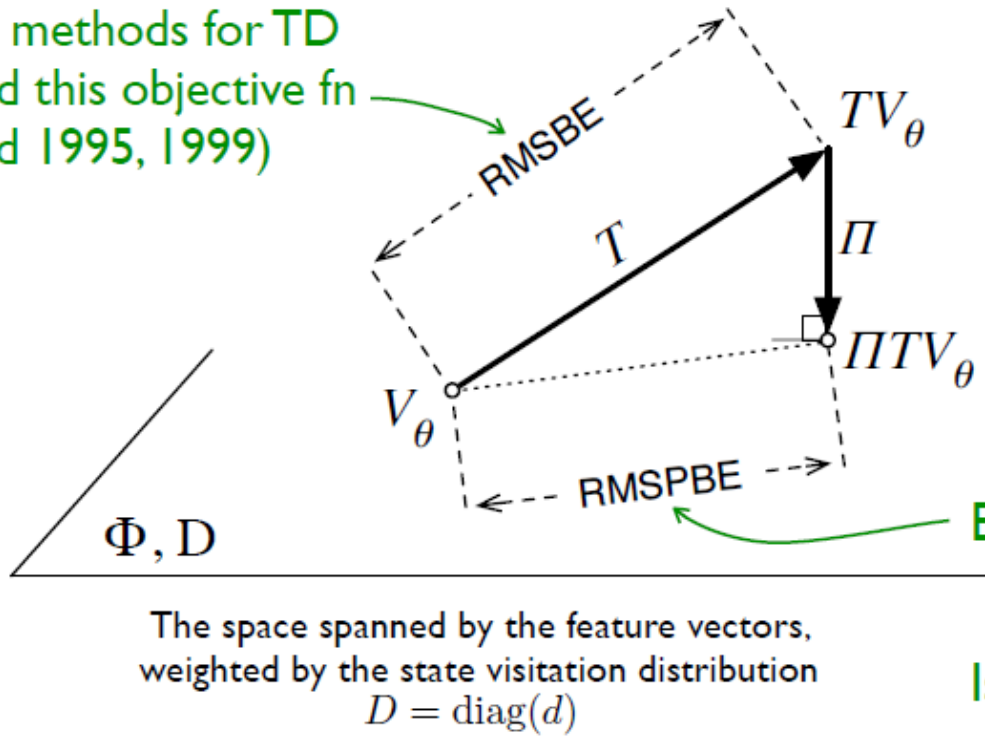
- Mean-Square Projected Bellman Error:

$$MSPBE(\theta) = ||V_\theta - \Pi TV_\theta||^2$$

Projected Bellman Error

Projected Bellman Error: $|V_\theta - \Pi T V_\theta|$

Previous work on gradient methods for TD minimized this objective fn (Baird 1995, 1999)



T takes you outside the space

Π projects you back into it

Better objective fn?

$V_\theta = \Pi T V_\theta$

Is the TD fix-point

The space spanned by the feature vectors, weighted by the state visitation distribution $D = \mathrm{diag}(d)$

Control 算法的收敛性

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

(✓) 表示在最优价值函数附近震荡

针对控制学习算法，大多数都能得到较好的策略，但是理论上只要存在函数近似，就都不是严格收敛的，比较常见的是在最优策略上下震荡，逐渐逼近然后突然来一次发散，再逐渐逼近等。使用非线性函数近似的效果要比线性函数要差很多

Table of Contents

批处理强化学习

- 梯度方法简单并且有效，但是并不能很有效地利用到样本数据
  - 递增算法都是基于数据流的，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
- 批处理 (batch) 方法希望找到一个最好的数据拟合方法
  - 批方法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符合这段时期内所有的数据。这里的训练数据集“块”相当于个体的一段经验

最小二乘法预测（Least Squares Prediction）

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

$$D = \{\langle s_1, v_1^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$$

- 什么样的参数 w 能得到最好的拟合结果？
- 最小二乘法通过最小化 $\hat{v}(s_t, \boldsymbol{w})$ 和真实值 v_t^π 之间的误差平方和，从而得到参数 $\boldsymbol{w}$

$$\begin{aligned} LS(\boldsymbol{w}) &= \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \boldsymbol{w}))^2 \\ &= E_D[(v^\pi - \hat{v}(s, \boldsymbol{w}))^2] \end{aligned}$$

Stochastic Gradient Descent with Experience Replay

- 经历重现（Experience Replay），把一段时期内的经历重新过一遍，更新参数
- 给定一段时期的、包含 $<$ 状态、价值 $>$ 的经历 D:

$$D = \{\langle s_1, v_1^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$$

- 重复:
  1. 从经历中去选出一个 $\langle s, v^\pi \rangle$

$$\langle s, v^\pi \rangle \sim D$$

  2. 应用随机梯度下降来更新参数:

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

- 这将收敛至针对这段经历最小平方差的参数:

$$\boldsymbol{w}^\pi = argmin_{\boldsymbol{w}} LS(\boldsymbol{w})$$

经历重现应用于 DQN 网络

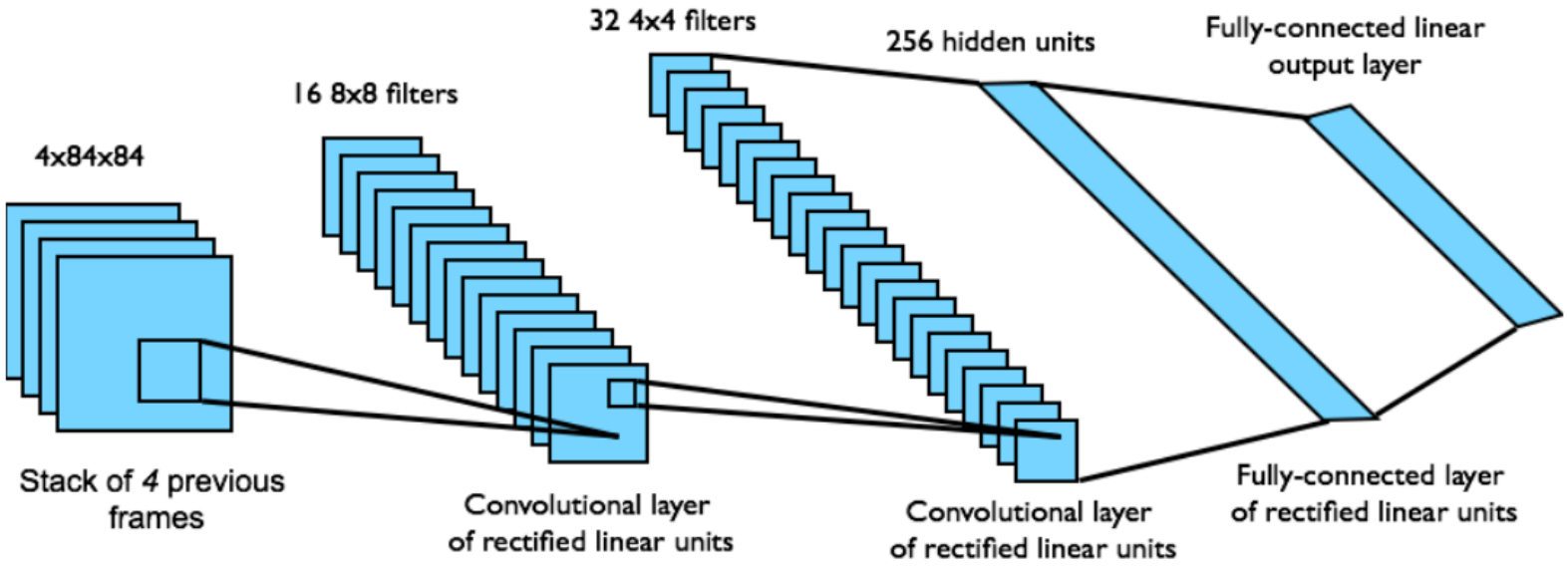
- TD 方法结合非线性的神经网络函数近似可能不会收敛，但 DQN(Deep Q-Networks) 使用经历重现和固定的 Q 目标值能做到收敛而且保持很好的鲁棒性
- DQN 算法的要点：
  1. 依据 ϵ-greedy 执行策略产生 t 时刻的行为
  2. 将大量经历数据（例如百万级的）以 $(s_t, a_t, r_{t+1}, s_{t+1})$ 存储在内存里，作为 D 大块
  3. 从 D 大块中随机抽取小块（例如 64 个样本数据）数据 (s, a, r, s')
  4. 维护两个神经网络 DQN1，DQN2，一个网络固定参数专门用来产生目标值，目标值相当于标签数据。另一个网络专门用来评估策略，更新参数。
  5. 优化关于 Q 网络和 Q 目标值之间的最小平方差：

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

  式中：w^- 在小块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 in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



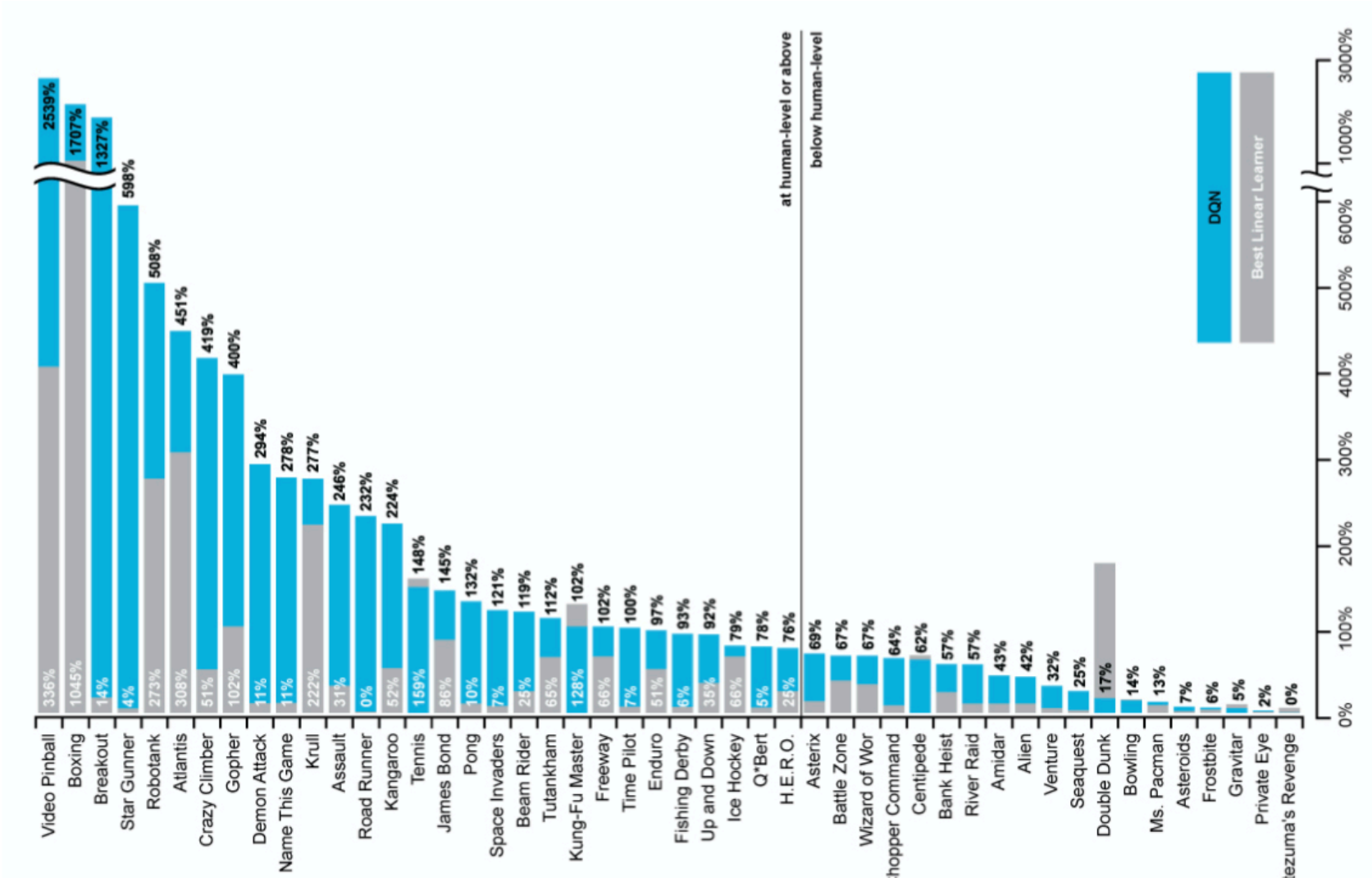
Network architecture and hyperparameters fixed across all games.

经历重现应用于 DQN 网络

- 随机采样打破了状态之间的联系；第二个神经网络会暂时冻结参数，我们从冻结参数的网络而不是从正在更新参数的网络中获取目标值，这样增加了算法的稳定性。经过一次批计算后，把冻结参数的网络换成更新的参数再次冻结产生新一次迭代时要用的目标值
- 下表比较了在 DQN 中有没有应用固定参数、以及有没有使用经历重现（批方法）两个条件时在 DQN 玩 Atari 5 款游戏中的表现，结果体现了这两个条件联合应用的优势：

	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

DQN Results in Atari



Network architecture and hyperparameters fixed across all games.

批方法的直接计算

- 经历重现（Experience Replay）使用能够找到最小平方差的解决方案，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：

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

然后直接求解参数

- 求解思路是逆向思维，假设已经找到这个参数，则它应该满足最小 $LS(\boldsymbol{w})$，即 $E_D[\Delta \boldsymbol{w}] = 0$，通过把 LS 展开，可以直接得到 $\boldsymbol{w}$

$$\begin{aligned}\Delta \boldsymbol{w} &= \alpha(v^\pi - \hat{v}(s, \boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{v}(s, \boldsymbol{w}) \\ &= \alpha(v^\pi - \hat{v}(s, \boldsymbol{w}))x(s)\end{aligned}$$

线性最小二乘法预测（Linear Least Squares Prediction）

- 当 $LS(\boldsymbol{w})$ 最小化时，其 expected update 一定为 0

$$E_D[\Delta \boldsymbol{w}] = 0$$

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

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

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

- 如果状态 s 有 N 个特征，那上面方法的运行时间为 $O(N^3)$；利用 Shermann-Morrison 公式（求解逆矩阵的一种方法），求解复杂度为 $O(N^2)$
- 求解该问题的难度与设计的特征数量多少有关，而与状态空间大小无关，因此适合应用与那些特征较少的问题

线性最小二乘法预测算法 (1)

- 在强化学习中，我们并不知道真实的值 V_t^π
- 但在实际中，我们可利用一些有误差或者偏差的方法去构建 V_t^π
  - LSMC least Squares Monte-Carlo 用回报 G_t

$$V_t^\pi \approx G_t$$

  - LSTD Least Squares Temporal-Difference 用 TD-Target $R_{t+1} + \gamma \hat{V}(S_{t+1}, \boldsymbol{w})$

$$V_t^\pi \approx R_{t+1} + \gamma \hat{V}(S_{t+1}, \boldsymbol{w})$$

  - LSTD(λ) Least Squares TD(λ) 用 G_t^λ

$$V_t^\pi \approx G_t^\lambda$$

- 上面的每种情况都可以直接求解

线性最小二乘法预测算法 (2)

- LSMC

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

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

- LSTD

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

$$\boldsymbol{w} = \left(\sum_{t=1}^{T} x(s_t)(x(s_t) - \gamma x(s_{t+1}))^\top \right)^{-1} \sum_{t=1}^{T} x(s_t)R_{t+1}$$

线性最小二乘法预测算法 (3)

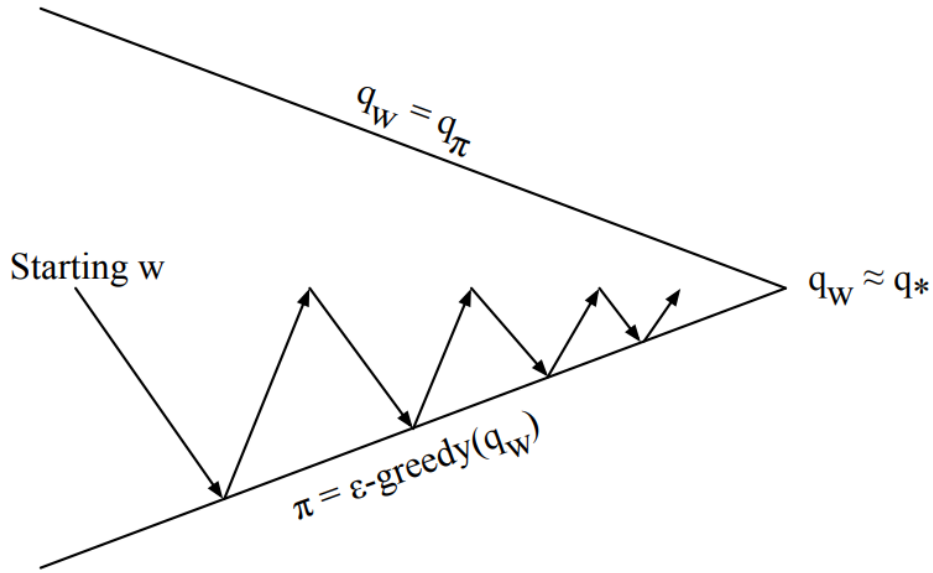
- LSTD(λ)

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})$$
$$E_t = \gamma \lambda E_{t-1} + x(S_t)$$
$$0 = \sum_{t=1}^{T} \alpha \delta_t E_t$$
$$\boldsymbol{w} = \left(\sum_{t=1}^{T} E_t (x(s_t) - \gamma x(s_{t+1}))^\top \right)^{-1} \sum_{t=1}^{T} E_t R_{t+1}$$

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	✓	✓	-

Least Squares Policy Iteration



- 策略评估：用最小二乘 Q-learning 去做策略评估
- 策略提升：采用 Greedy 策略

最小二乘行动值函数估计

- action-value 函数估计 $q_\pi(s, a)$
- 针对行为值函数的线性近似

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

- 用最小二乘法最小化 $q_\pi(s, a)$ 和 $\hat{q}(s, a, \boldsymbol{w})$ 之间的误差
- 用策略 π 生成样本数据 $\langle(\text{state}, \text{action}), \text{value}\rangle$

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

最小二乘法控制

- 对于 policy evaluation，我们想有效地利用所有的数据
- 对于 control 问题，我们还想提升策略
- 样本数据通常是由多种策略生成的
- 所以 evaluate $q_\pi(s, a)$ 使用 off-policy learning
- 我们采用 Q-learning 的思想:
  - 用旧的 policy 生成样本

$$S_t, A_t, R_{t+1}, S_{t+1} \sim \pi_{old}$$

  - 用新的策略选择不同的后继动作 $A' = \pi_{new}(S_{t+1})$
  - 用 $R_{t+1} + \gamma \hat{q}(s_{t+1}, A', \boldsymbol{w})$ 更新 $\hat{q}(s_t, A_t, \boldsymbol{w})$

最小二乘 Q-learning

- 考虑下面的线性 Q-learning 更新

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

- LSTDQ 算法：solve for total update = zero

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

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

最小二乘策略迭代算法

- 下面的算法用 LSTDQ 作为 policy evalution
- 该算法用不同的 policy 重复地重新计算样本 D

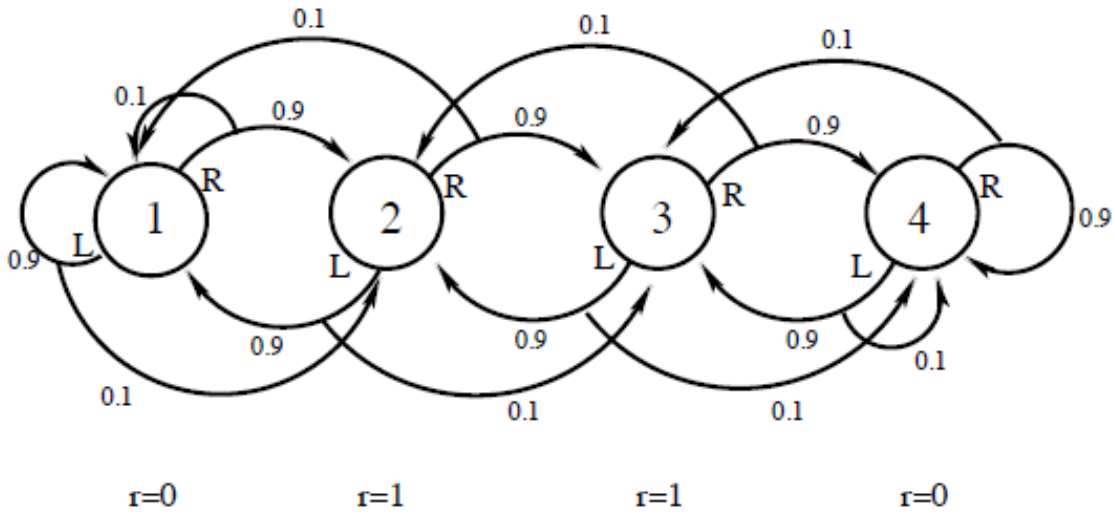
function LSPI-TD$(\mathcal{D}, \pi_0)$
  $\pi' \leftarrow \pi_0$
  repeat
    $\pi \leftarrow \pi'$
    $Q \leftarrow$ **LSTDQ**$(\pi, \mathcal{D})$
    for all $s \in \mathcal{S}$ **do**
      $\pi'(s) \leftarrow \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s, a)$
    end for
  until $(\pi \approx \pi')$
  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	✓	(✓)	-

(✓) = chatters around near-optimal value function

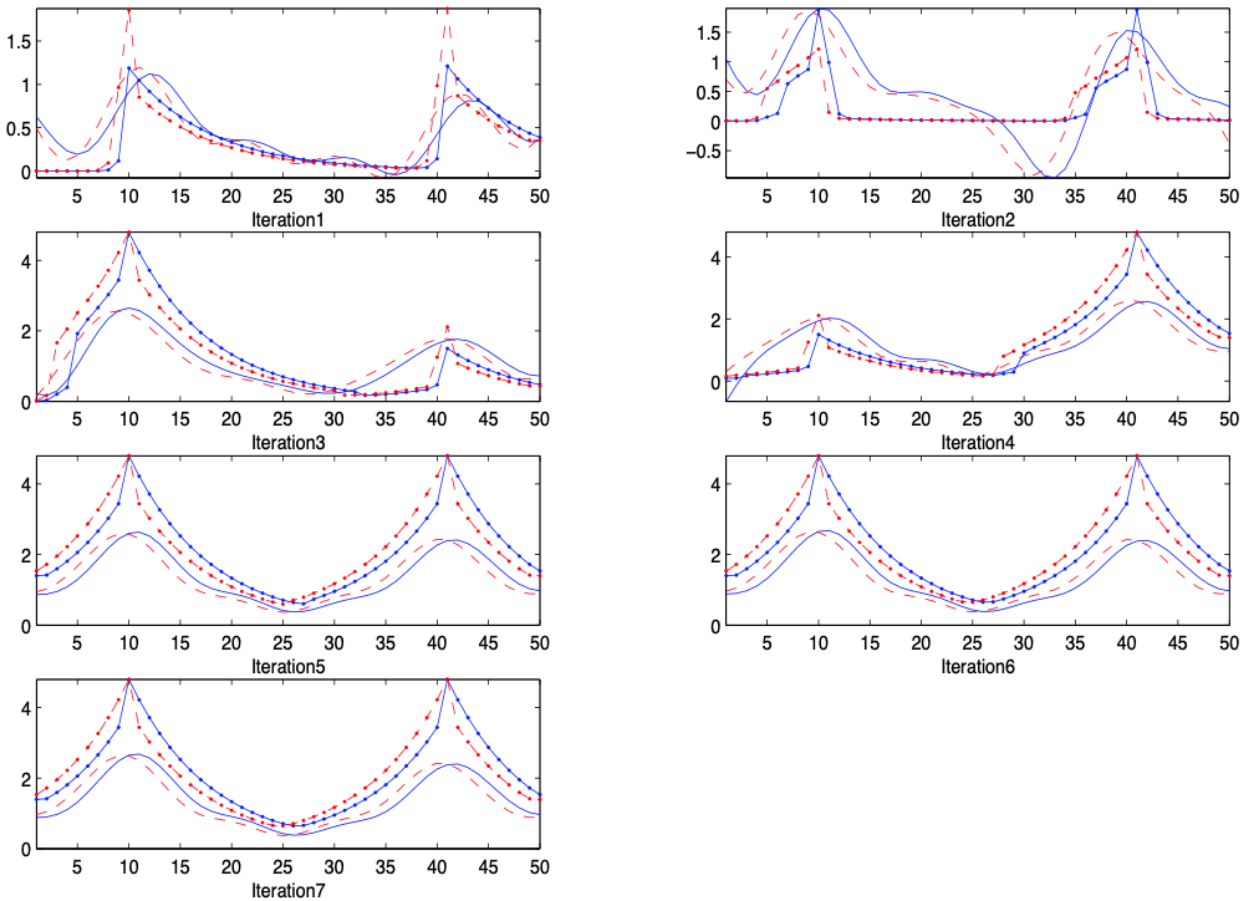
Chain Walk Example



- Chain Walk: 随机行走问题的状态空间扩展为 50 个
- 所有状态可分为两类：状态 10 和 41 是两个好的状态，获得奖励 1；其他状态均获得奖励 0。
- 最优策略是：当个体处在 1-9 这 9 个状态时采取向右移步的行为，用 R(1-9) 表示，在其他状态时的最优行为分别是：L(10-25)，R(26-41)，和 L(42-50)
- 算法使用的特征是：针对每一个行为（左或右）设计 10 个均匀分布的高斯分布状态特征
- Experience: 10,000 steps from random walk policy

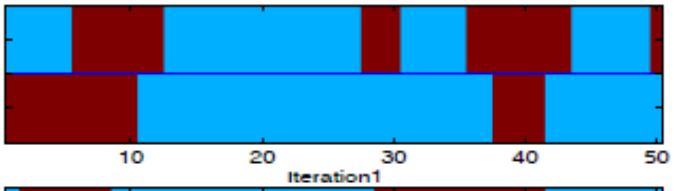
LSPI in Chain Walk: Action-Value Function

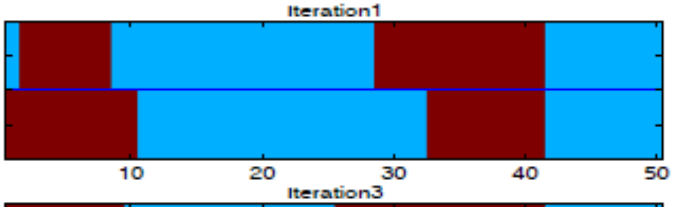
红、蓝色分别代表左右不同的行为对应的值函数

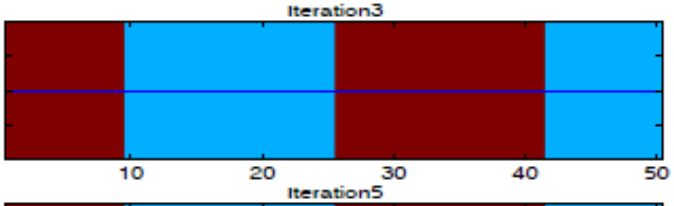


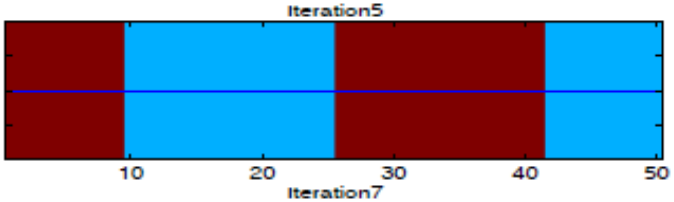
LSPI in Chain Walk: Policy

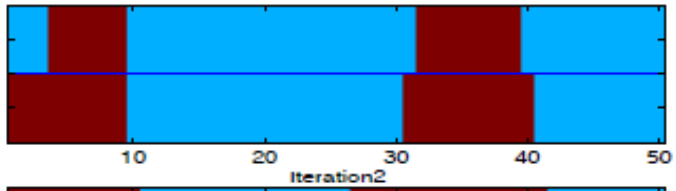
红、蓝色分别代表左右不同的行为

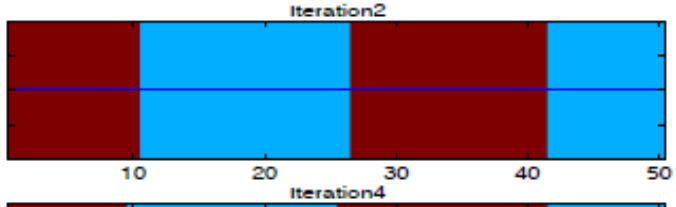


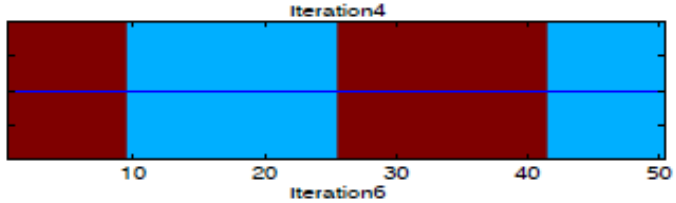












小结

- 在处理大规模状态问题时，很自然地提出了一种函数估计的方法（VFA）
- VFA，我们主要集中于梯度的方法，所以介绍 gradient descent
- 然后介绍 increment method，主要就是随机梯度方法
  - prediction：MC, TD(0), TD(λ)（主要估计 V 值）
  - control：MC, TD(0), TD(λ)（主要估计 Q 值）
  - 注意这些方法的收敛性
- 而 batch method，主要利用了最小二乘法，它可以一步求解
  - prediction：LSMC, LSTD(0), LSTD(λ)（主要估计 V 值）
  - control：LSPI（主要估计 Q 值）
  - 注意这些方法的收敛性