多智能体系统与强化学习

主讲人：高阳、杨林、杨天培

https://reinforcement-learning-2025.github.io/

第三讲：函数估计

走向大规模问题

高　阳

大 纲

值函数估计

值函数估计的更新

线性值函数估计中的状态表征

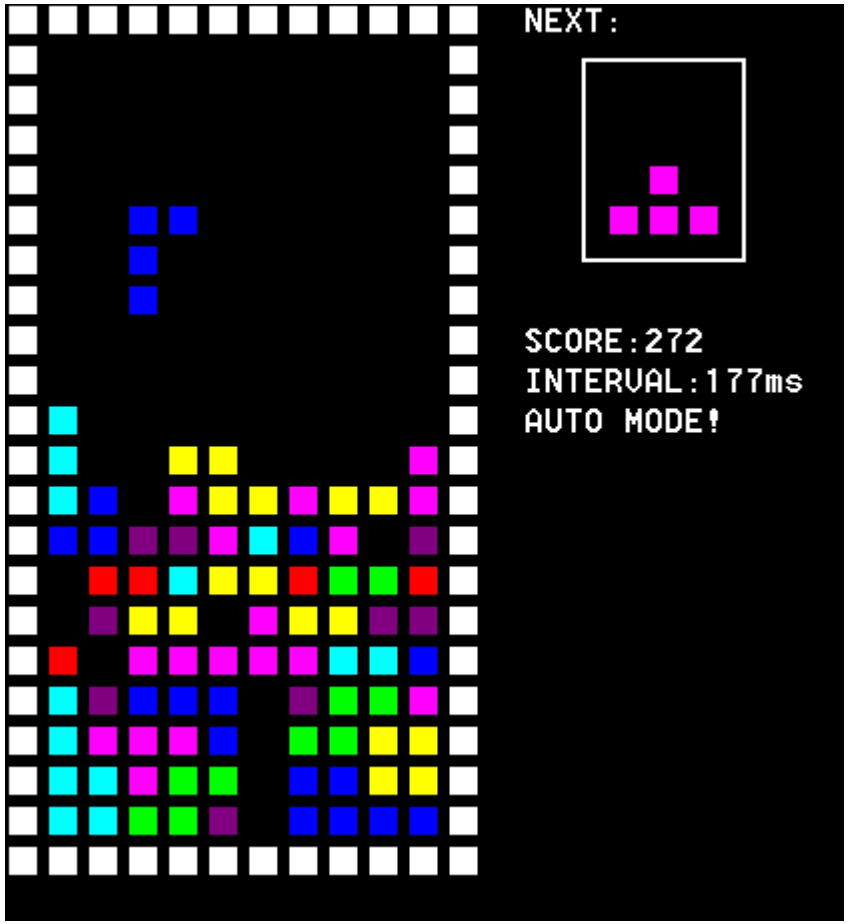
大 纲

值函数估计

值函数估计的更新

线性值函数估计中的状态表征

俄罗斯方块游戏



- Tetris游戏
  - ✓ 1984年，由俄罗斯程序员Alexey Pazhitnov发明
  - ✓ 标准游戏桌面：行宽10，列高20
  - ✓ 块的形状：S、Z、L、I、O、T
- RL问题视角
  - ✓ 状态空间$2^{200}*6$
  - ✓ 动作：平移、旋转
  - ✓ 即时奖赏：每消除一行得正分

用强化学习求解俄罗斯方块游戏

Playing Tetris with Deep Reinforcement Learning

Matt Stevens
mslf@stanford.edu

Sabeek Pradhan
sabeekp@stanford.edu

Abstract

We used deep reinforcement learning to train an AI to play tetris using an approach similar to [7]. We use a convolutional neural network to estimate a Q function that describes the best action to take at each game state. This approach failed to converge when directly applied to predicting individual actions with no help from heuristics. However, we implemented many features that improved convergence. By grouping actions together, we allowed the Q network to focus on learning good game configurations instead of how to move pieces. By using transfer learning from a heuristic model we were able to greatly improve performance, having already learned relevant features about Tetris and optimal piece configurations. And by using prioritized sweeping, we were able to reduce the some of the inherent instability in learning to estimate single actions. Although we did not surpass explicitly featurized agents in terms of performance, we demonstrated the basic ability of a neural network to learn to play Tetris, and with greater training time we might have been able to beat even those benchmarks.

1. Introduction

The primary motivation for this project was a fun application of neural networks. Tetris is a well-known game that is loved and hated by many. Recent results from the team at DeepMind have shown that neural networks can have remarkable performance at game playing, using a minimal amount of prior information about the game. However, there has not yet been a successful attempt to use a convolutional neural network to learn to play Tetris. We hope to fill this gap.

Furthermore, research into solving control problems with neural networks and minimal explicit featurization may have applications outside of just video games, such as in self-driving cars or robotics.

Tetris is a game that involves placing pieces on top of each other so that they fill in a rectangular grid. A common human approach to this problem involves looking for open spaces that match the shape of the current piece in play. This is a ultimately a task in visual pattern detection and object recognition, so we believe that a convolutional neural network is a natural approach to solving this problem. Tetris also involves a fair degree of strategy, and recent advances in deep reinforcement learning have shown that convolutional neural networks can be trained to learn strategy.

2. Related Work

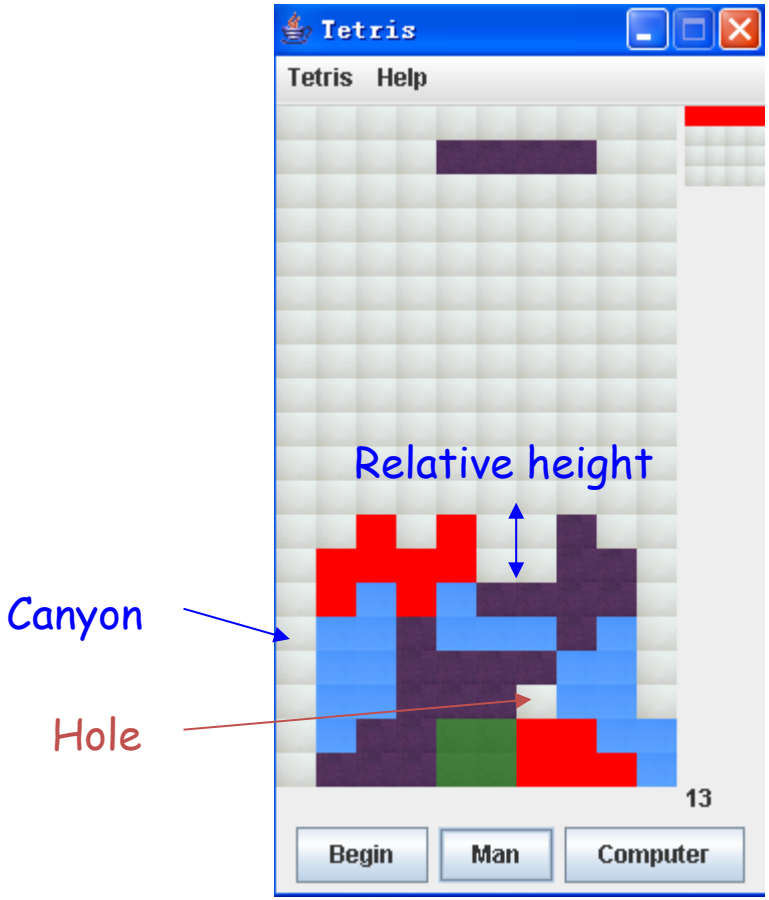
One of the seminal works in the field of deep reinforcement paper was DeepMind's 2013 paper, Playing Atari with Deep Reinforcement Learning [6]. This paper and their 2015 follow-up [7] served as our primary inspiration, both for the general idea applying computer vision to reinforcement learning for video games and for ideas on how to proceed with our model architecture. Their results are extremely impressive and state-of-the-art, with [7] achieving superhuman performance on over half the games tested and falling short primarily on games that require long-term strategic planning and have very sparse rewards, such as Montezuma's Revenge.

Several of the techniques that [6], [7], and our paper use have much longer histories. For example, experience replay, which stores the game-playing agent's experiences to be trained long after the original policy has been abandoned, was introduced in [4]. RMSProp, an optimization technique that naturally anneals learning rates down over time and focuses its learning rates on dimensions with more consistent and thus informative gradients, was first introduced in [9]. [6] and [7] recommended but did not implement prioritized sweeping as a way to select saved examples to train on; we implemented this in our paper. Prioritized sweeping, which trains the model on the examples with the largest error, was introduced in [8]; a more computationally efficient version, which we did not implement but would consider as an extension, was introduced in [1]. Finally, one optimization method we experimented with but that [6] and [7] did not mention was Adadelta. Adadelta, which is less hyperparameter-dependent than other optimization methods and anneals effective learning rates down based on how much a given dimension has already been updated, was introduced in [11].

Finally, it is worth noting that previous attempts to build

关系强化学习(Relational RL)

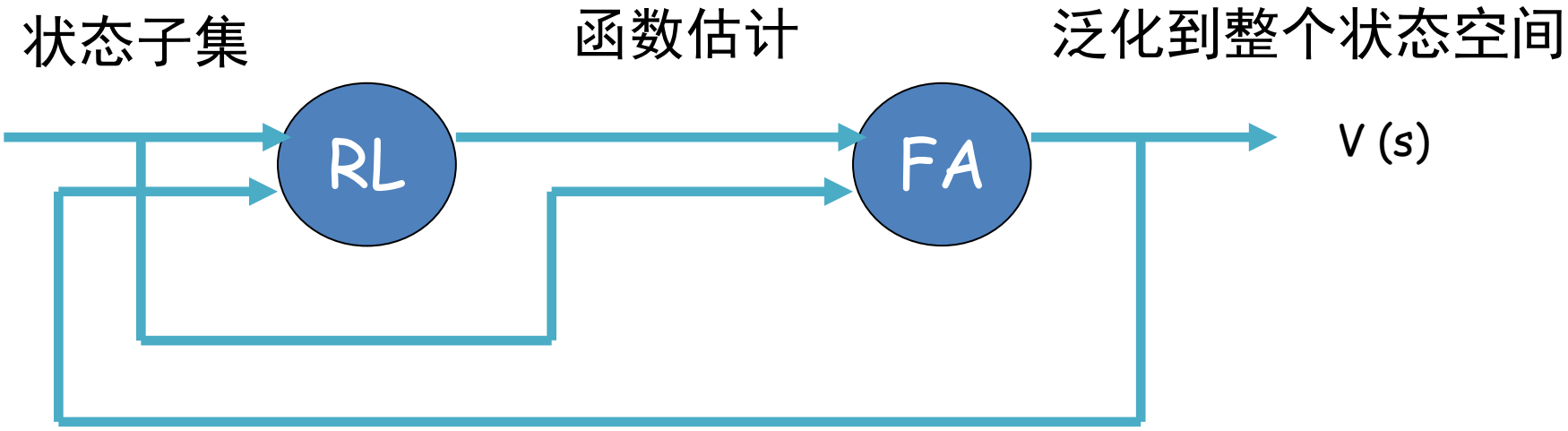
- 属性
  - Height of wall (max, avg, min)
  - Number of Holes
  - Height difference adjacent cols
  - Canyon (width, height)
  - ...
- 宏动作
  - Fits
  - Increasesheight, ...
  - Number of deleted lines
  - ...



值函数估计

- 简单RL问题
  - ✓ 值函数存储以查找表(lookup table)方式
  - ✓ 对应算法称之为值表型强化学习算法
- 复杂RL问题
  - ✓ 在时间步t，值函数$V(s_t)$，依赖于参数向量$\boldsymbol{w}_t$
  - ✓ $\boldsymbol{w}$可以是显式的，如关系强化学习
  - ✓ $\boldsymbol{w}$可以是隐式的，如神经网络强化学习

强化学习中的值函数估计



$$V_0, M(V_0), \Gamma(M(V_0)), M(\Gamma(M(V_0))), \Gamma(M(\Gamma(M(V_0)))), \ldots$$

注意：存在两个逼近过程

大 纲

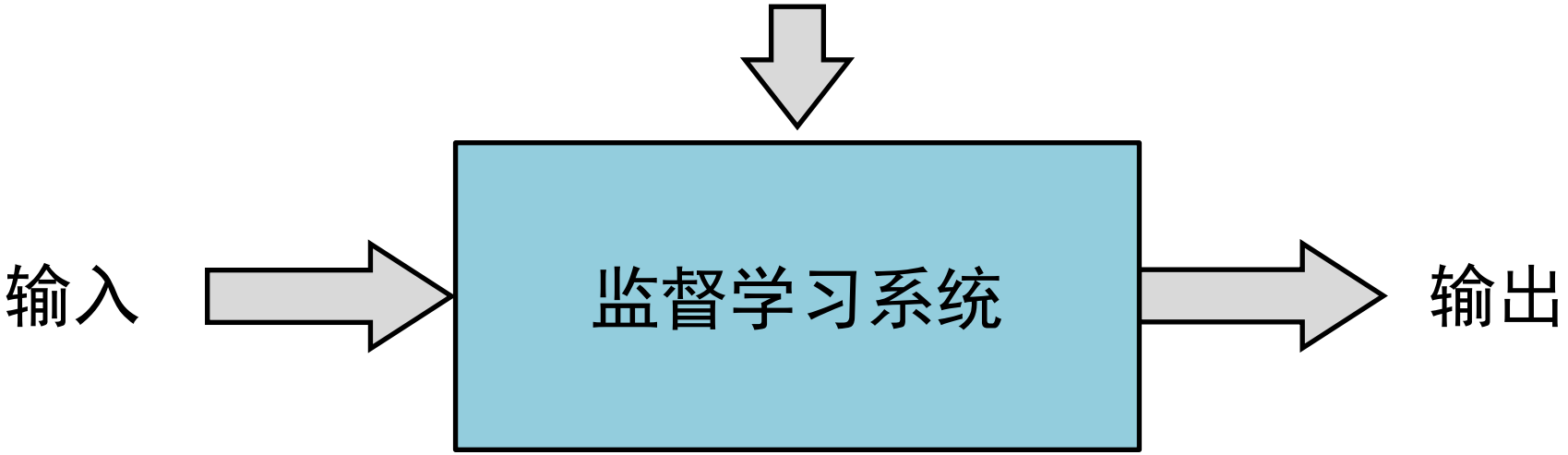
值函数估计

值函数估计的更新

线性值函数估计中的状态表征

监督学习视角

实际输出=期望（目标输出）



训练样本 = {输入，目标输出}

误差 = (目标输出—实际输出)

值表强化学习

- 值表型强化学习更新公式

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

监督学习训练视角下

$$\{\text{Description of } s_t,\ r_{t+1} + \gamma V(s_{t+1})\}$$

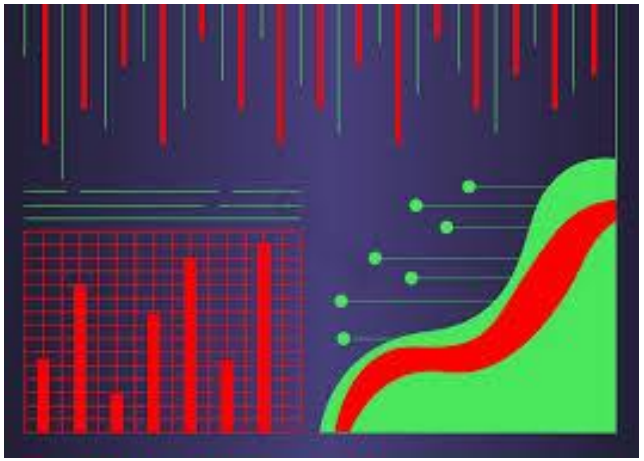
输入　　目标输出

其他值函数估计方法

- 值函数估计器
  - ✓ 特征的线性组合
  - ✓ 决策树
  - ✓ 神经网络
  - ✓ 核函数
  - ✓ ……

但强化学习算法仍然需要特殊考虑：

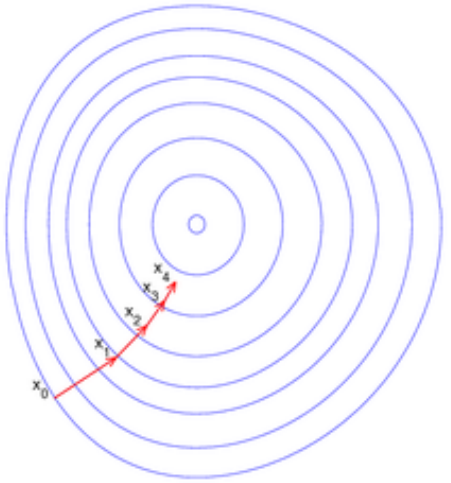
- ✓ 学习过程非稳定（non-stationary）
- ✓ 数据非独立同分布（non-iid）



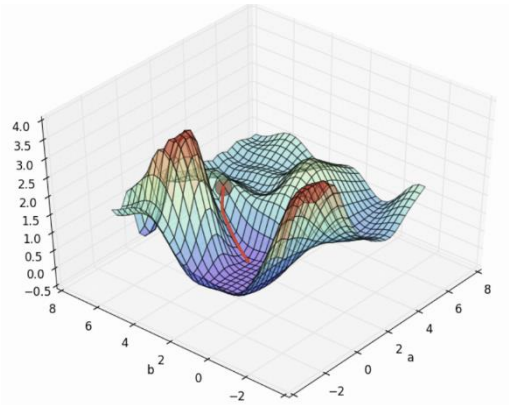
梯度下降方法(前修课程：最优化、机器学习)

- 存在一个带有参数$\boldsymbol{w}$的可微函数$J(\boldsymbol{w})$
- 定义$J(\boldsymbol{w})$的梯度如下：

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



- **目标**：找到$J(\boldsymbol{w})$的一个局部最小值
- 一个局部下降最快的方向——**负梯度**



$$\Delta \boldsymbol{w} = -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \quad \alpha\text{为一个步长参数}$$

基于随机梯度下降的值函数估计

- **目标：**寻找$\boldsymbol{w}$最小化值函数**估计值**与**真实值**之间的均方误差

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}\left[\left(V^{\pi}(s) - \hat{V}(s,\boldsymbol{w})\right)^2\right]$$

真实值　　估计值

- 误差减少的方向

$$-\frac{1}{2}\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \mathbb{E}_{\pi}\left[\left(V^{\pi}(s) - \hat{V}(s,\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{V}(s,\boldsymbol{w})\right]$$

- 采样梯度下降

$$\Delta\boldsymbol{w} = -\frac{1}{2}\alpha\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \alpha\left(V^{\pi}(s) - \hat{V}(s,\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{V}(s,\boldsymbol{w})$$

线性值函数估计

- **线性值函数**可由特征与参数线性组合而成

$$\hat{V}(s, \boldsymbol{w}) = \boldsymbol{x}(s)^{\top}\boldsymbol{w} = \sum_{i=1}^{n} w_i x_i(s)$$

- 目标函数可进一步表示为

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(V^{\pi}(s) - \boldsymbol{x}(s)^{\top}\boldsymbol{w})^2]$$

- 鉴于随机梯度方法可以收敛到最优解，于是更新公式如下

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

$$= \boldsymbol{w} + \alpha \left(V^{\pi}(s) - \hat{V}(s, w)\right) \boldsymbol{x}(s)$$

步长　　预测误差　　特征值

线性值函数估计

- **值表**可看作线性值函数估计的一种特殊形式
- 值表特征可以表示

$$\boldsymbol{x}^{\text{table}}(s) = \begin{bmatrix} 1(s = s_1) \\ \vdots \\ 1(s = s_n) \end{bmatrix}$$

指示函数

- 值表函数估计可以进一步表示

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

线性值函数估计

- 随机梯度优化假设真实值函数V^{π}可知，但在强化学习中，仅有奖励信号。因此，需要找到目标值替代真实值函数

  - ✓ **蒙特卡罗方法**（MC），使用回报G_t

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

  - ✓ **时差学习**TD(0)，使用目标值$R_{t+1} + \gamma \hat{V}(s_{t+1}, \boldsymbol{w})$

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \hat{V}(s_{t+1}, \boldsymbol{w}) - \hat{V}(s_t, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{V}(s_t, \boldsymbol{w})$$

  - ✓ **时差学习**TD(λ)，使用λ −回报G_t^{λ}

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

蒙特卡罗值函数估计

- 回报G_t是关于真实值函数的带噪音无偏采样(unbiased sampling)
- 在在线收集的训练数据上运用监督学习对价值函数进行预测

$$\langle s_1, G_1 \rangle, \langle s_2, G_2 \rangle, \dots, \langle s_T, G_T \rangle$$

- 对于每个数据样本

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left(G_t - \hat{V}(s_t, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{V}(s_t, \boldsymbol{w})$$

$$= \boldsymbol{w} + \alpha \left(G_t - \hat{V}(s_t, \boldsymbol{w}) \right) \boldsymbol{x}(s_t)$$

- 蒙特卡罗估计至少能收敛到一个局部最优解
- 在值函数为线性的情况下可以收敛到全局最优

时差TD(0)值函数估计

- 目标值$R_{t+1}+\gamma\hat{V}(s_{t+1},\boldsymbol{w})$是关于真实值函数的有偏采样
- 在“训练数据”上运用监督学习对价值函数进行预测

$$\langle s_1, R_2+\gamma\hat{V}(s_2,\boldsymbol{w})\rangle, \langle s_2, R_3+\gamma\hat{V}(s_3,\boldsymbol{w})\rangle, \dots, \langle s_{T-1}, r_T\rangle$$

- 对于每个数据样本

$$\boldsymbol{w}\leftarrow\boldsymbol{w}+\alpha\left(R_{t+1}+\gamma\hat{V}(s_{t+1},\boldsymbol{w})-\hat{V}(s_t,\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{V}(s_t,\boldsymbol{w})$$

$$=\boldsymbol{w}+\alpha\left(R_{t+1}+\gamma\hat{V}(s_{t+1},\boldsymbol{w})-\hat{V}(s_t,\boldsymbol{w})\right)\boldsymbol{x}(s_t)$$

- 线性情况下时差学习(接近)收敛到全局最优解

时差TD(λ)值函数估计

- λ –回报G_t^λ同样是真实值函数的有偏采样
- 在“训练数据”上运用监督学习对价值函数进行预测

$$\langle s_1, G_1^\lambda \rangle, \langle s_2, G_2^\lambda \rangle, \dots, \langle s_{T-1}, G_{T-1}^\lambda \rangle$$

- 前向视角线性TD(λ)

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left(G_t^\lambda - \hat{V}(s_t, \boldsymbol{w}) \right) \boldsymbol{x}(s_t)$$

- 后向视角线性TD(λ)

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

$$E_t = \gamma \lambda E_{t-1} + \boldsymbol{x}(s_t)$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \delta_t E_t$$

大 纲

值函数估计

值函数估计的更新

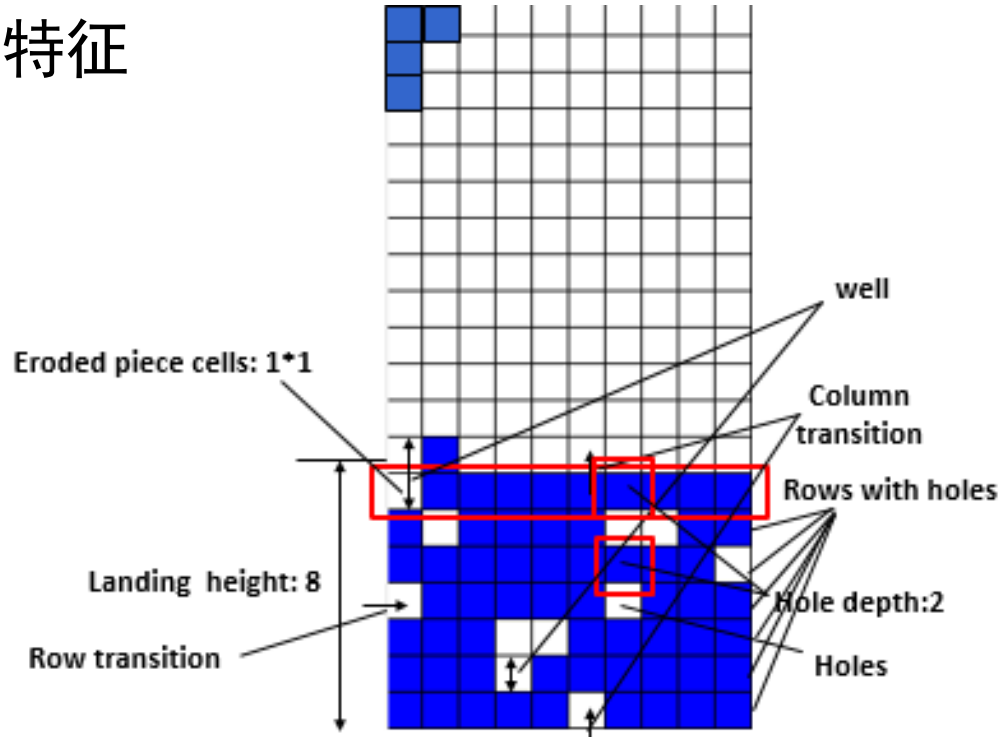
线性值函数估计中的状态表征

线性值函数估计的状态表征

- 状态特征表示一般形式

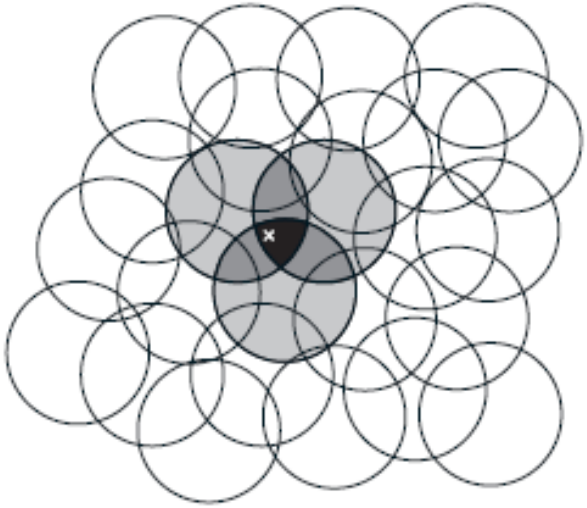
$$\boldsymbol{x}(s) = \begin{bmatrix} \boldsymbol{x}_1(s) \\ \vdots \\ \boldsymbol{x}_k(s) \end{bmatrix}$$

- 专家定义俄罗斯方块(Tetris)特征
  - ✓ 空洞数
  - ✓ 下落高度
  - ✓ 井深度
  - ✓ 行转移数
  - …… ……

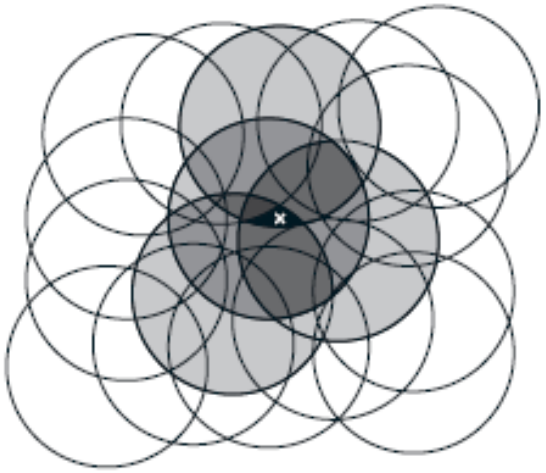


线性值函数估计的状态表征

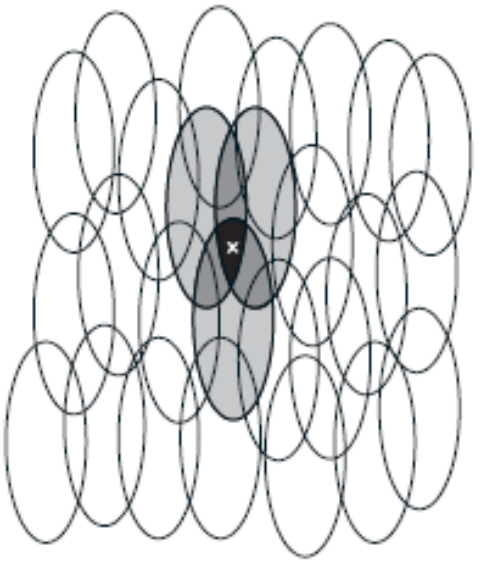
- 状态特征表示的其他方式
  - ✓ 粗编码(Coarse Coding)：状态重叠性质的特征



Narrow generalization



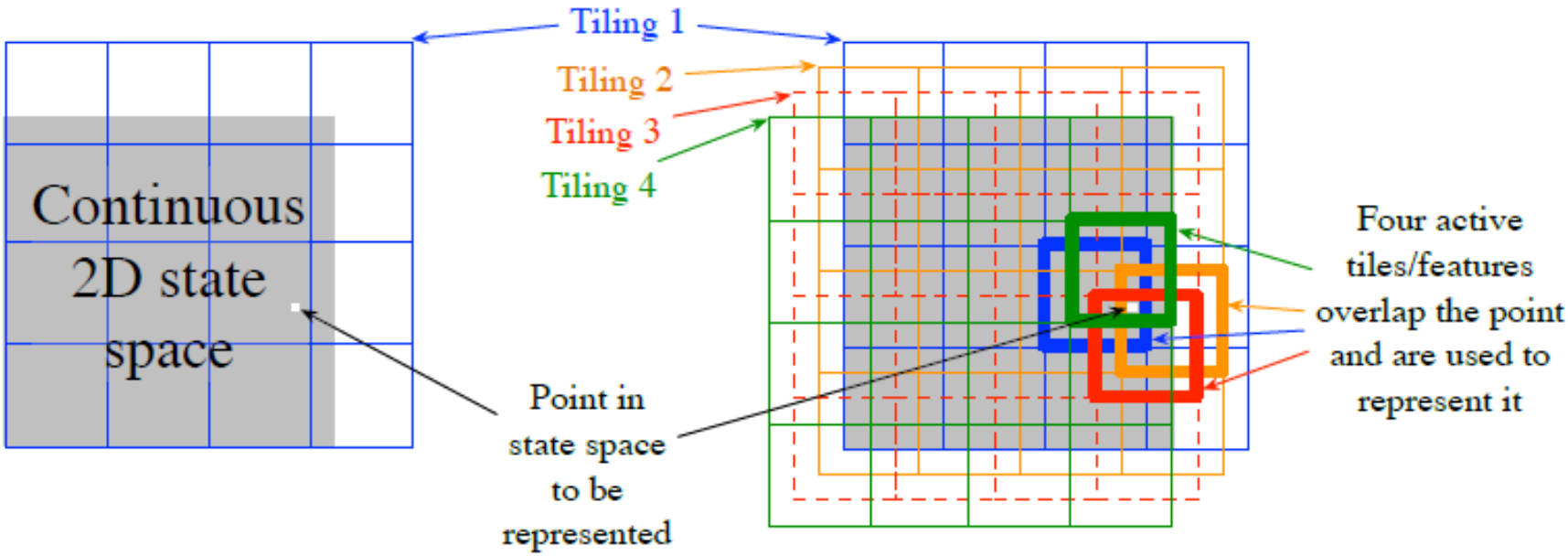
Broad generalization



Asymmetric generalization

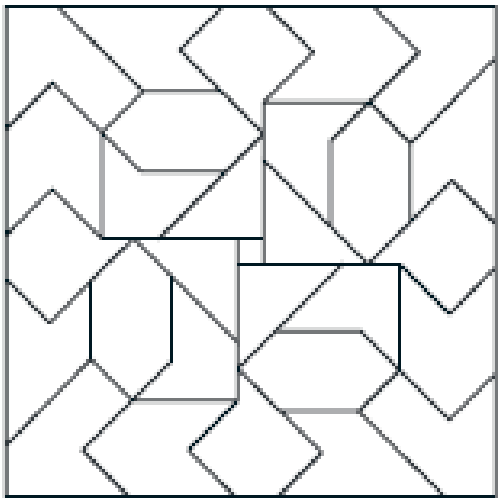
线性值函数估计的状态表征

- 状态特征表示的其他方式
  - ✓ 块编码(Tile Coding)：连续空间的粗编码

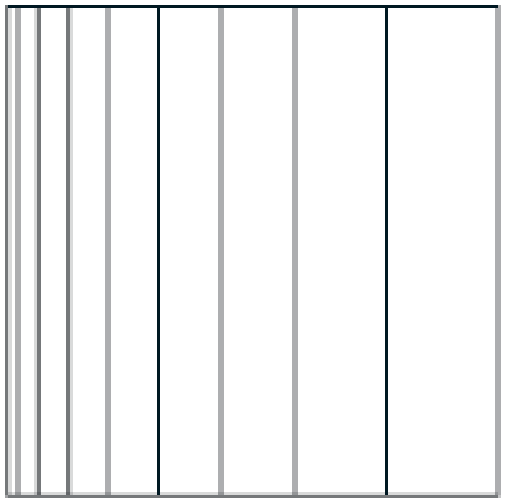


线性值函数估计的状态表征

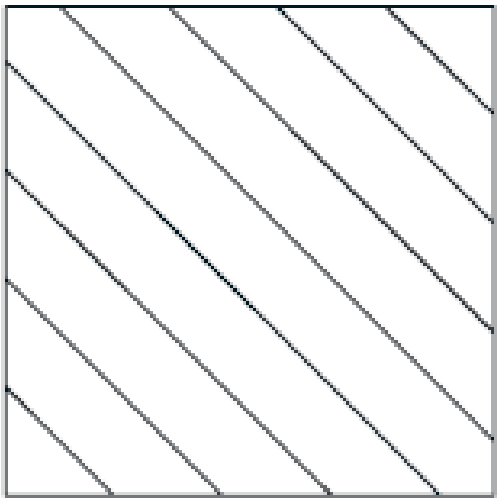
- 状态特征表示的其他方式
  - ✓ 块编码(Tile Coding)：存在不同的划分方式



Irregular



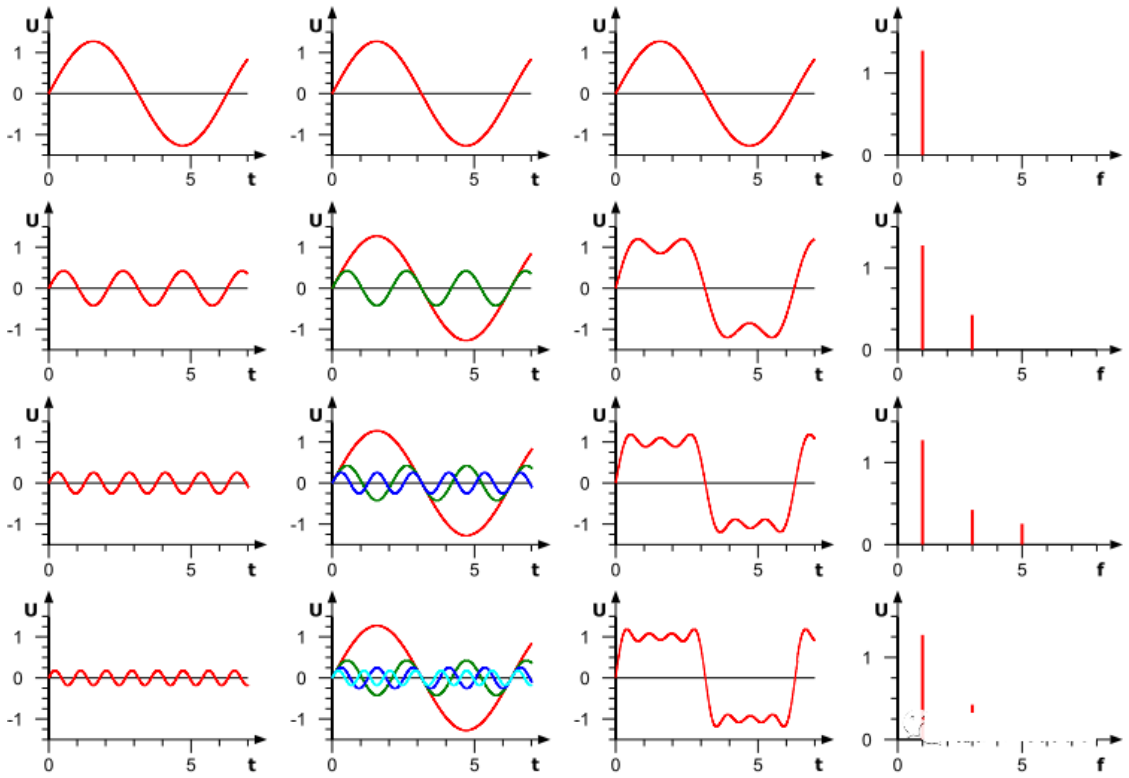
Log stripes



Diagonal stripes

线性值函数估计的状态表征

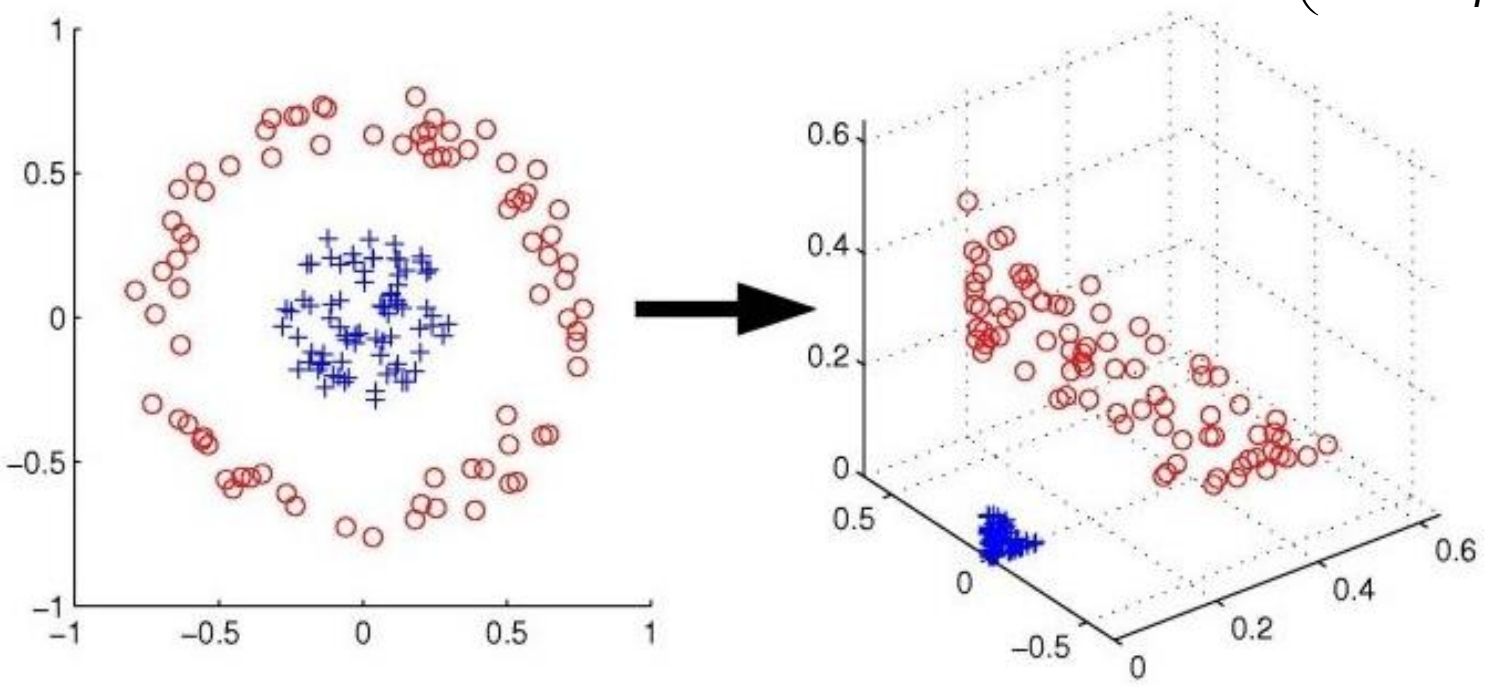
- 状态特征表示的其他方式
  - ✓ 傅立叶基：可由不同振幅，不同相位弦波叠加



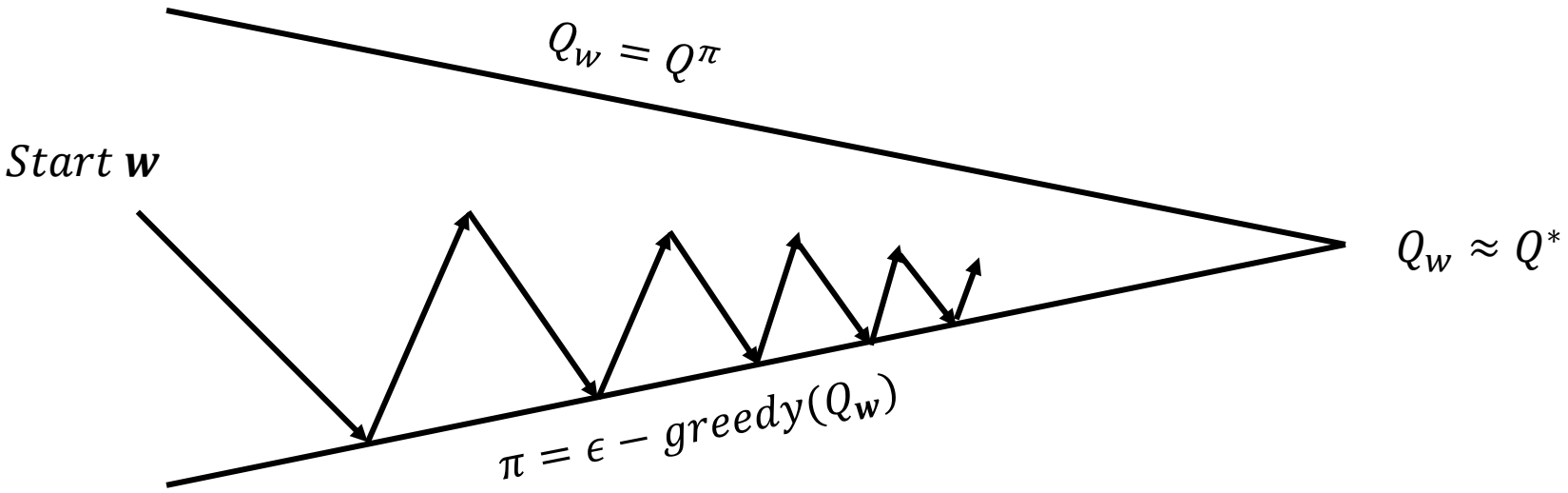
线性值函数估计的状态表征

- 状态特征表示的其他方式
  - ✓ 核函数：从低维不可分到高维可分

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



值函数估计用作控制



- 策略评估：估计状态-动作值函数，$Q(\cdot,\cdot,\boldsymbol{w}) \approx Q^\pi$
- 策略改进：$\epsilon - greedy$策略探索改进

状态–动作值函数估计

- 同样，对动作-状态值函数进行估计

$$\hat{Q}(s,a,w) \approx Q^{\pi}(s,a)$$

- 最小均方误差

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}\left[\left(Q^{\pi}(s,a) - \hat{Q}(s,a,\boldsymbol{w})\right)^2\right]$$

- 在单个样本上进行随机梯度下降

$$\Delta \boldsymbol{w} = -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \alpha\left(Q^{\pi}(s,a) - \hat{Q}(s,a,\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{Q}(s,a,\boldsymbol{w})$$

线性状态–动作值函数估计

- 通过特征向量表示状态-动作

$$\boldsymbol{x}(s,a) = \begin{bmatrix} \boldsymbol{x}_1(s,a) \\ \vdots \\ \boldsymbol{x}_k(s,a) \end{bmatrix}$$

- 线性状态–动作值函数可以表示如下

$$\hat{Q}(s,a,\boldsymbol{w}) = \boldsymbol{x}(s,a)^{\top}\boldsymbol{w} = \sum_{i=1}^{n} w_1 x_1(s,a)$$

- 基于随机梯度的更新如下

$$\Delta \boldsymbol{w} = \alpha \left(Q^{\pi}(s,a) - \hat{Q}(s,a,\boldsymbol{w}) \right) \boldsymbol{x}(s,a)$$

线性状态-动作值函数估计

- 与预测算法一致，需要使用目标值替换未知真实值$Q^{\pi}(s,a)$
  - ✓ **MC**，使用回报G_t

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

  - ✓ **TD(0)**，使用目标值$r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}, w)$

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

  - ✓ **前向视角TD(λ)**，使用λ −回报G_t^{λ}

$$\Delta \boldsymbol{w} = \alpha \left(G_t^{\lambda} - \hat{Q}(s_t, a_t, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t, \boldsymbol{w})$$

  - ✓ **后向视角TD(λ)**，类似

$$\delta_t = r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}, \boldsymbol{w}) - \hat{Q}(s_t, a_t, \boldsymbol{w}), E_t = \gamma \lambda E_{t-1} + \boldsymbol{x}(s_t)$$

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

思考和讨论

1. 特征与值函数估计器
2. 强化学习中的值函数估计更新与监督学习中有什么不同
3. 写出利用线性值函数的强化学习算法

谢 谢 ！