# Reinforcement Learning: An Introduction notebook

黎雷蕾

2017 年 12 月 13 日

# 目录

**9 On-policy Prediction with Approximation 2**

9.1 Value-function Approximation . . . . . 2
9.2 The Prediction Objective (MSVE) . . . . . 2
9.3 Stochastic-gradient and Semi-gradient Methods . . . . . 3
9.4 Linear Methods . . . . . 5
9.5 Feature Construction for Linear Methods . . . . . 7
9.5.1 Polynomials(多项式) . . . . . 7
9.5.2 Fourier Basis(傅立叶基) . . . . . 7
9.5.3 Coarse Coding . . . . . 7
9.5.4 Tile Coding . . . . . 8
9.5.5 Radial Basis Functions(径向基函数) . . . . . 9
9.6 Nonlinear Function Approximation: Artificial Neural Networks 9
9.7 Least-Squares TD, LSTD . . . . . 9
9.8 Memory-based Function Approximation . . . . . 11
9.9 Kernel-based Function Approximation . . . . . 11

# Chapter 9

# On-policy Prediction with Approximation

## 9.1 Value-function Approximation

明白:$s \mapsto g$,

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

在不同的算法中:

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

## 9.2 The Prediction Objective (MSVE)

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

MSVE) 就可以写作：

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

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

## 9.3 Stochastic-gradient and Semi-gradient Methods

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

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

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

其中权重向量的偏导向量可以表示为：

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

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

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

该算法可以写为：

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

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

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

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

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

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

## 9.4 Linear Methods

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

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

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

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

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

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

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

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

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

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

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

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

将其拓展到 semi-gradient TD(n) 算法，则：

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

算法可以写为：

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

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

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

## 9.5 Feature Construction for Linear Methods

### 9.5.1 Polynomials(多项式)

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

该思想可以描述为:

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

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

### 9.5.2 Fourier Basis(傅立叶基)

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

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

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

对应的步长参数:

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

### 9.5.3 Coarse Coding

Coarse coding 通常用来处理有许多重叠特征的状态。

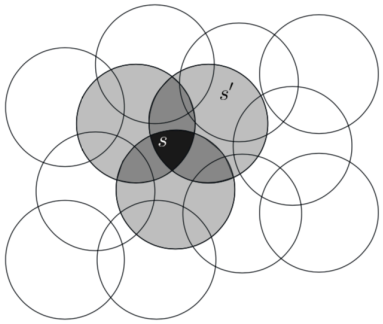


Figure 9.6: Coarse coding. Generalization from state $s$ to state $s'$ depends on the number of their features whose receptive fields (in this case, circles) overlap. These states have one feature in common, so there will be slight generalization between them.

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

### 9.5.4 Tile Coding

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

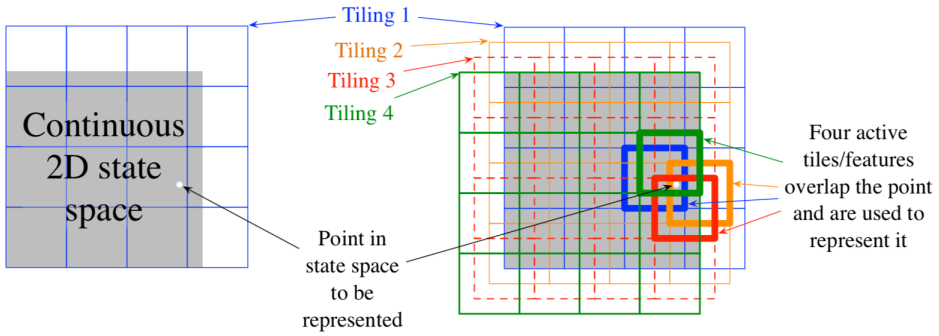


Figure 9.9: Multiple, overlapping grid-tilings on a limited two-dimensional space. These tilings are offset from one another by a uniform amount in each dimension.

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

### 9.5.5 Radial Basis Functions(径向基函数)

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

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

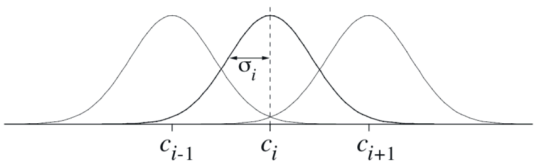


Figure 9.13: One-dimensional radial basis functions.

## 9.6 Nonlinear Function Approximation: Artificial Neural Networks

这节主要是介绍神经网络的，没有什么干货。

## 9.7 Least-Squares TD, LSTD

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

假设:

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

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

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

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

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

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

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

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

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

## 9.8 Memory-based Function Approximation

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

## 9.9 Kernel-based Function Approximation