Subject: Westlake University, Reinforce Learning, Lecture 8, Value Function Approximation

Date: from February 12, 2025 to February 14, 2025

Contents

Lecture 8, Value Function Approximation

Bilibili:Lecture 8, Value Function Approximation

Introduction

在前几讲中我们一直使用表格(tabular method) 的方式来表示和更新 state value 与 action value，这种方法直观但无法应对大规模或连续的状态/动作空间。本节开始正式引入函数逼近(Function Approximation) 的思想，用参数化函数替代表格，从而实现对 value function 的高效表示与泛化.

本文主要内容包括：

1. 函数逼近的基本动机：用少量参数表示大量状态价值，解决存储、泛化等问题.
2. 函数选择：线性函数、特征向量方法，以及神经网络作为非线性逼近器.
3. 与 TD learning 结合：将函数逼近与时序差分学习结合，构建优化问题，使用梯度下降或随机梯度下降更新参数.
4. 理论分析：证明了线性函数逼近下TD 学习的收敛性，提出投影贝尔曼误差(Projected Bellman Error) 的概念，并给出收敛解的数学证明.
5. 扩展算法：
   (a) Sarsa with Function Approximation
   (b) Q-learning with Function Approximation
   (c) Deep Q-learning (DQN)，引入了 Target Network 和 Experience Replay 技巧
6. 附录：详细证明收敛性、投影贝尔曼误差等数学细节，以及马尔可夫过程的平稳分布(stationary distribution).

Motivating examples: curve fitting

Table

在本节以前，state value 与 action value 都是以表格(table)的形式给出，例如:

	a_1	a_2	a_3	a_4	a_5
s_1	$q_\pi(s_1,a_1)$	$q_\pi(s_1,a_2)$	$q_\pi(s_1,a_3)$	$q_\pi(s_1,a_4)$	$q_\pi(s_1,a_5)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_9	$q_\pi(s_9,a_1)$	$q_\pi(s_9,a_2)$	$q_\pi(s_9,a_3)$	$q_\pi(s_9,a_4)$	$q_\pi(s_9,a_5)$

Table 1: Table of action values

s_1	s_2	s_3	s_4	$\cdots$	s_9
$v_\pi(s_1)$	$v_\pi(s_2)$	$v_\pi(s_3)$	$v_\pi(s_4)$	$\cdots$	$v_\pi(s_9)$

Table 2: Table of state values

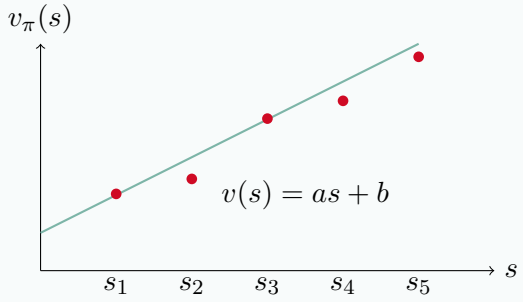
使用表格形式处理数据的优点在于其易于理解和分析，但是其缺点在于难以处理大型或者连续的 state / action space，原因有三点：

1. **Storage 1:** 当空间是离散时，若离散值较多则需要大量存储空间
2. **Storage 2:** 当空间是连续时需要进行(网格)离散化，当网格稀疏时不能很好地近似原空间，当网格稠密即节点过多时又面临存储问题
3. **Generalization:** 表格更新只能逐点进行，不能够影响周围未被访问到的值

Curve fitting

为解决上述问题，可以使用函数近似(**function approximation**)，其思想是使用较简单的函数去近似 action / state value 与其自变量 (s, a) / s 间的函数关系，例如：

- 考虑一维的 state $s_1, \dots, s_{|\mathcal{S}|}$，记 $\mathcal{S}$ 为 state space，$|\mathcal{S}|$ 为所有的状态个数
- 上述 state 对应的 state value 为 $v_\pi(s_1), \dots, v_\pi(s_{|\mathcal{S}|})$，其中 π 为给定的 policy
- 假设 $|\mathcal{S}|$ 非常大，我们希望使用一个简单的曲线拟合这些 (state, state value) 对，即 $(s, v_\pi(s))$，这样只需要存储该曲线较少的参数.



最简单的方式就是使用直线进行拟合（如上图所示），其可以写为如下形式：

$$\hat{v}(s, w) = as + b = \underbrace{[s, 1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{w} = \phi^T(s)w \tag{1}$$

其中 w 为 parameter vector (参数向量)，$\phi(s)$ 为 s 的 feature vector (特征向量).

function approximation 与 tabular method 的不同之处在于

1. **Store:** 原本我们需要存储 $|\mathcal{S}|$ 个 state value，但是现在只需要存储两个参数 a 和 b
2. **Retrieve:** 每当我们需要使用到 s 的 state value 时，需要计算 $\phi^T(s)w$，
3. **Accuracy:** 由于这只是一个拟合/近似（approximation），因此会引入误差，故我们牺牲了精度来提高存储效率.
4. **Update:** table 中直接逐点更新即可，而前者必须更新 w 以间接更新 state value.
5. **Generalization ability:** 前者更新 w 也会影响其他一些状态的值，即使尚未访问这些状态，故一个 state 的 experience sample 可以帮助估计其他一些 state 的值.

Summary

1. **Idea:** 使用参数化的函数去近似 state / action value: $\hat{v}(s, w) \approx v_\pi(s) / \hat{p}(s, a, w) \approx p_\pi(s, a, w)$，其中 $w \in \mathbb{R}^m$ 为 parameter vector.

2. **Advantage**

   (a) **Storage:** 由于 w 的维数较 $|\mathcal{S}|$ 少很多，因此具有存储上的巨大优势

   (b) **Generalization:** 在使用 table 形式存储数据时，当某个 state / action value 发生改变时其他的值并不会进行更新，因此若不能将全部的数据访问完，就会造成大量 state value 未更新；但若使用函数拟合，改变了某一点的 state / action value 就会改变 parameter vector，这样全部的 state / action value 都会被更新.

Selection of function approximators

在进行拟合时，一个关键问题是选择什么样的拟合函数 $\hat{v}(s,w)$，这主要有两种方式：

1. **Linear function:** $\hat{v}(s,w)=\phi^T(s)w$，其中 $\hat{v}(s,w)$ 为 feature vector. 值得注意的是，虽然 $\hat{v}(s,w)$ 对于 $\phi^T(s)$ 是非线性的，但是对于参数 w 来说仍是线性的. $\hat{v}(s,w)$ 可以选取 $\phi(\cdot)$ 为多项式基底、Fourier 基底等，选择更复杂的基底可以减小误差但相应地也增加了参数存储量，例如使用二阶曲线拟合：

$$\hat{v}(s,w)=as^2+bs+c=\underbrace{[s^2,s,1]}_{\phi^T(s)}\underbrace{\begin{bmatrix}a\\b\\c\end{bmatrix}}_{w}=\phi^T(s)w. \tag{2}$$

2. **Neural networks:** 特征向量的选择需要很多先验知识，若无任何先验知识，一个被广泛采用的解决方案是使用神经网络作为非线性函数逼近器，即神经网络的参数为 θ，输入神经网络的为 state，输出为非线性的 $\hat{v}(s,\theta)$.

Linear function approximator

选用线性函数逼近有如下优缺点：

1. **Disadvantages:** 选取合适的特征向量(feature vectors)需要先验知识.

2. **Advantages:**

   (a) 线性情形下的理论分析较非线性情形时更简单，目前已经有了很多的理论结果，且可以帮助理解非线性情形.

   (b) 拟合能力虽不如非线性，但已经较好. 且线性逼近可以将 tabular method 与 function approximation 统一，因为 tabular method 实际上是线性函数逼近的一个特殊形式.

当我们将 feature vector 选取为单位向量时，即

$$\phi(s)=e_s\in\mathbb{R}^{|S|}$$

其中 e_s 只有第 s 个元素为1，其余为0.

那么此时就有

$$\hat{v}(s,w)=e_s^Tw=w(s)$$

其中 $w(s)$ 为 w 的第 s 个元素.

此时将 $\phi(s_t) = e_s$ 带入 TD-linear 的算法1 linear case(TD linear)中得到

$$\begin{aligned} w_{t+1} &= w_t + \alpha_t \left[r_{t+1} + \gamma \phi^T(s_{t+1}) w_t - \phi^T(s_t) w_t \right] \phi(s_t) \\ &= w_t + \alpha_t \left(r_{t+1} + \gamma w_t(s_{t+1}) - w_t(s_t) \right) e_{s_t} \end{aligned}$$

由于 e_{s_t} 只有第 s_t 个元素非零，因此得到

$$w_{t+1}(s_t) = w_t(s_t) + \alpha_t \left(r_{t+1} + \gamma w_t(s_{t+1}) - w_t(s_t) \right)$$

将 w 替换为 v 就与 tabular TD algprithm 形式一模一样.

TD learning based on function approximation

Introduction

本节将 function approximation 与上一节中的 TD Learning 算法结合，以估计相应的 state value / action value / optimal policy. 将介绍如下几个部分：

1. 将 function approximation 形式化为一个优化问题(optimization problem)
2. 如何求解该优化问题，即优化算法(optimization algorithm)
3. 特征向量(feature vectors)的选择问题
4. 理论分析(theoretical analysis)

Objective function

Objective function

现在，在 state evaluation 任务中我们的目标是找到最优的参数向量 w，使其能够对于所有的 s 都能最好地近似 $v_\pi(s)$，为此我们需要两个步骤：

1. 定义目标函数(objective function)
2. 使用合适的算法去优化该目标函数

我们直接给出目标函数的定义如下：

$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2] \tag{3}$$

我们的目标是最小化 $J(w)$ 以找到 optimal w。注意这里的 S 是一个随机变量，需要取期望，因此其分布的选择很重要，主要有如下两种方式：

1. **uniform distribution:** 即取到每个 state 的概率是一样的，都是 $1/|\mathcal{S}|$，此时目标函数可写为：

$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2] = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (v_\pi(s) - \hat{v}(s, w))^2 \tag{4}$$

   平均分布假设所有的 state 同等重要，但实际上接近目标的状态应该是更重要的，需要分配更多的权重，同时有一些 state 很少被访问，应该被分配更少的权重.

2. **stationary / steady-state / limiting distribution:** 描述了马尔可夫决策过程的长期行为(**long-run behavior**)，即在给定 policy 下不断环境交互，达到平稳状态时每个 state 被访问次数的概率分布，将其记为 $\{d_\pi(s)\}_{s\in\mathcal{S}}$，其中 $d_\pi(s)\geqslant 0, \sum_{s\in\mathcal{S}} d_\pi(s) = 1$。这样得到的目标函数为

$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))^2] = \sum_{s\in\mathcal{S}} d_\pi(s)(v_\pi(s) - \hat{v}(s,w))^2. \tag{5}$$

分布 $d_\pi(s)$ 的获取并不简单，一种方式是通过仿真实验近似:

$$d_\pi(s) \approx \frac{n_\pi(s)}{\sum_{s'\in\mathcal{S}} n_\pi(s')}$$

实际上不需要进行仿真实验也可以在理论上得到 $d_\pi(s)$ 的理论值，其需要需要研究 state 概率转移矩阵 P_π，详细分析见附录A.

Optimization algorithms

Gradient decent

为最小化目标函数 $J(w)$，我们可以使用梯度下降算法:

$$w_{k+1} = w_k - \alpha_k \nabla_{*w} J(w_k)$$

其中梯度为

$$\begin{aligned}\nabla_w J(w) &= \nabla_w \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))^2] = \mathbb{E}[\nabla_w (v_\pi(S) - \hat{v}(S,w))^2] \\ &= -2\mathbb{E}[(v_\pi(S) - \hat{v}(S,w))\nabla_w \hat{v}(S,w)]\end{aligned}$$

由此得到相应的梯度下降(GD)和随机梯度下降(SGD)算法为:

$$\begin{aligned}&\text{GD:} \quad w_{k+1} = w_k + 2\alpha_k \mathbb{E}[(v_\pi(S) - \hat{v}(S,w_k))\nabla_w \hat{v}(S,w_k)], \\ &\text{SGD:} \quad w_{t+1} = w_t + 2\alpha_t \left(v_\pi(s_t) - \hat{v}(s_t,w_t)\right) \nabla_w \hat{v}(s_t,w_t).\end{aligned} \tag{6}$$

但是可以看到在进行梯度下降时，已经使用了我们需要求解的目标 v_π，因此我们需要将其替换才能运行算法，常见的有两种方法:

1. **Monte Carlo:** 使用从 s_t 出发的一个 episode 的 discounted return g_t 代替 $v_\pi(s_t)$:

$$w_{t+1} = w_t + \alpha_t(g_t - \hat{v}(s_t,w_t))\nabla_w \hat{v}(s_t,w_t)$$

2. **TD learning:** 使用 $r_{t+1} + \gamma\hat{v}(s_{t+1},w_t)$ 代替 $v_\pi(s_t)$:

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma\hat{v}(s_{t+1},w_t) - \hat{v}(s_t,w_t)\right] \nabla_w \hat{v}(s_t,w_t) \tag{7}$$

在 TD learning case 中如果使用线性函数逼近 $\hat{v}(s,w)$，即

$$\hat{v}(s,w) = \phi^T(s)w, \tag{8}$$

那么就得到了 TD-Linear 算法(详见算法1):

$$\text{TD linear: } w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma\phi^T(s_{t+1})w_t - \phi^T(s_t)w_t\right] \phi(s_t). \tag{9}$$

Algorithm 1 TD Learning with Function Approximation

Initialization: A function $\hat{v}(s,w)$ that is differentiable in w. Initial parameter w_0.
Aim: Approximate the true state values of a given policy π.
for each episode generated following the policy π **do**
  for each step (s_t, r_{t+1}, s_{t+1}) **do**
    In the general case:

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)\right] \nabla_w \hat{v}(s_t, w_t)$$

    In the linear case(TD linear):

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \phi^T(s_{t+1}) w_t - \phi^T(s_t) w_t\right] \phi(s_t)$$

  end for
end for

Summary of the story

在将 TD learning 算法融合 function approximation 的故事中，我们首先定义了优化的目标函数，将问题转化为优化问题，再通过梯度下降对问题进行求解，但在算法中需要通过 function approximation 对目标函数进行替换，最终构成了完整的算法1.

objective function → gradient decent → replace the true value function by approximation

Theoretical analysis

Convergence analysis

由于非线性的分析较为复杂，因此本部分近考虑线性情形，即 TD-Linear (9).
在分析 TD learning 算法(9)之前，首先分析其确定性形式：

$$w_{t+1} = w_t + \alpha_t \mathbb{E}\left[\left(r_{t+1} + \gamma \phi^T(s_{t+1}) w_t - \phi^T(s_t) w_t\right) \phi(s_t)\right], \tag{10}$$

其中 s_t 的分布为 stationary distribution d_π. 分析确定性形式基于以下两点考虑：

1. 确定性算法的收敛性更容易（尽管不是平凡的）分析.
2. 确定性算法(10)的收敛性蕴含随机 TD 算法(9)的收敛性.

为方便分析，首先定义如下两个矩阵：

$$\Phi = \begin{bmatrix} \vdots \\ \phi^T(s) \\ \vdots \end{bmatrix} \in \mathbb{R}^{n \times m}, \quad D = \begin{bmatrix} \ddots & & \\ & d_\pi(s) & \\ & & \ddots \end{bmatrix} \in \mathbb{R}^{n \times n}, \tag{11}$$

其中 Φ 为特征向量矩阵，包含了 n 个 state 的特征向量 $\phi(s) \in \mathbb{R}^m$，D 为对角阵，元素 d_{ii} 为 state s_i 的 stationary distribution $d_\pi(s)$.

Lemma 1. 式*(10)*中的期望可以写为

$$\mathbb{E}\left[\left(r_{t+1} + \gamma \phi^T\left(s_{t+1}\right) w_t - \phi^T\left(s_t\right) w_t\right) \phi\left(s_t\right)\right] = b - A w_t, \tag{12}$$

其中 $A \triangleq \Phi^T D\left(I-\gamma P_\pi\right)\Phi \in \mathbb{R}^{m\times m}$, $b \triangleq \Phi^T D r_\pi \in \mathbb{R}^m$.

Lemma 1 的证明可见附录B.1. 这样算法((10))就可以写为：

$$w_{t+1} = w_t + \alpha_t(b - Aw_t). \tag{13}$$

Convergence target: 在分析收敛性之前我们首先需要找到收敛的目标，因此首先假设 $w_t \xrightarrow{t\to\infty} w^*$，从而 $w^* = w^* + \alpha_\infty(b - Aw^*)$，因此得到

$$w^* = A^{-1}b. \tag{14}$$

为保证式(14)是有意义的，有几点说明：

1. A 可逆. 事实上 A 不仅可逆(invertible)，而且是正定(positive definite)的，即 $x^TAx > 0, \forall x \in \mathbb{R}^{\dim A}, x \neq \mathbf{0}$. 证明详见附录B.2.
2. $w^* = A^{-1}b$ 实际上是最小化投影贝尔曼误差(projected Bellman error)的最优解，详细说明见下一小节.
3. 在 tabular method 情形下，当 $\dim w = n = |\mathcal{S}|$，$\phi(s) = [0,\dots,1,\dots,0]^T$时，有

$$w^* = A^{-1}b = v_\pi. \tag{15}$$

说明要学习的参数向量实际上是真实 state value，与表格 TD算法是TD-Linear算法的特例这一事实是一致的. 式(15)的证明见附录B.3.

Convergence analysis: 在假设收敛的前提下得到收敛的目标 w^* 后，可以证明得到

$$w_t \xrightarrow{t\to\infty} w^* = A^{-1}b. \tag{16}$$

证明详见附录B.4.

Projected Bellman error

上一小节中提到 TD-Linear 算法收敛的解 $w^* = A^{-1}b$ 是是使投影贝尔曼误差(**projected Bellman error**)最小化的最优解，下面进行详细说明.

首先回顾一下目标函数的选择：

1. **Objective function 1: True value error**

$$J_E(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))^2] = \|\hat{v}(w) - v_\pi\|_D^2 \tag{17}$$

其中 D 为 S 对应的分布，$\|x\|_D^2 = x^TDx = \|D^{1/2}x\|_2^2$. 由于 v_π 未知，因此为获得可实现的算法还需考虑其他的目标函数.

2. **Objective function 2: Bellman error** 由于 v_π 满足 Bellman equation $v_\pi = r_\pi + \gamma P_\pi v_\pi$，所以期待 $\hat{v}(w)$ 尽可能满足方程 $\hat{v}(w) = r_\pi + \gamma P_\pi \hat{v}(w)$.

$$J_{BE}(w) = \|\hat{v}(w) - (r_\pi + \gamma P_\pi \hat{v}(w))\|_D^2 \triangleq \|\hat{v}(w) - T_\pi(\hat{v}(w))\|_D^2 \tag{18}$$

其中 $T_\pi(x) \triangleq r_\pi + \gamma P_\pi x$，称为 Bellman operator. 最小化 Bellman error 是一个标准的最小二乘问题.

3. **Objective function 3: Projected Bellman error** 由于使用的逼近函数可能出于函数结构原因无法使得 $\hat{v}(w) = r_\pi + \gamma P_\pi \hat{v}(w)$，因此可以使用投影矩阵 M 将 $T_\pi(\hat{v}(w))$ 投影到 $\hat{v}$ 的空间上使目标函数可以为 0.

$$J_{PBE}(w) = \|\hat{v}(w) - MT_\pi(\hat{v}(w))\|_D^2$$

其中 M 为正交投影矩阵(orthogonal projection matrix).

事实上算法(7)的目的是最小化 projected Bellman error J_{PBE} 而非 J_E 和 J_{BE}. 此处仅考虑线性情形 $\hat{v}(w) = \Phi w$，Φ 的 range space 为所有可能的线性近似的集合. 此时正交投影矩阵 M 为

$$M = \Phi(\Phi^T D \Phi)^{-1} \Phi^T D \in \mathbb{R}^{n \times n}.$$

由于 $\hat{v}(w)$ 和 $MT_\pi(\hat{v}(w))$ 均在 Φ 的 range space 中，因此能够找到 w 使得 J_{PBE} 最小化为 0. 同时有下面的结论(证明见附录B.5):

Theorem 1. $w^* = A^{-1}b = \arg\min_w J_{PBE}(w)$

TD algorithm 实际在最小化 J_{PBE} 而非 J_E，因此将 J_E 作为目标函数会使得估计值 $\hat{v}$ 与真实值 v_π 有差异. 同样地这里考虑线性情形 $\hat{v}(w^*) = \Phi w^*$，有如下结论(证明详见附录B.6).

Theorem 2. 在线性情形 $\hat{v}(w^*) = \Phi w^*$ 下，估计值与真实值 v_π 满足

$$\|\hat{v}(w^*) - v_\pi\|_D = \|\Phi w^* - v_\pi\|_D \leqslant \frac{1}{1-\gamma} \min_w \|\hat{v}(w) - v_\pi\|_D = \frac{1}{1-\gamma} \min_w \sqrt{J_E(w)}. \quad (19)$$

其中 γ 为 *discounted rate.*

Note 1. 虽然 ***Theorem 2*** 说明估计值与真实值差异的上界被 J_E 控制，但是实际这个 *upper bound* 很松，尤其是当 γ 很接近 *1* 时，因此主要具有理论意义.

Least-squares TD(LSTD)

暂略，可详见 textbook *Mathematical foundations of reinforcement learning*.

Sarsa with function approximation

Sarsa with function approximation

Sarsa 与之前的算法唯一的不同就在于是在使用 $\hat{q}(s,a,w)$ 估计 action value $q(s,a)$ 而不是使用 $\hat{v}(s,w)$ 来估计 state value $v(s)$，因此非常自然地可以得到使用了 function approximation 的 Sarsa 算法为:

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t) \quad (20)$$

式(20)实际上仍然在做 policy evaluation，将其与 policy improvement 结合就得到如下完整算法:

Algorithm 2 Sarsa with Function Approximation

Aim: Search a policy that can lead the agent to the target from an initial state-action pair (s_0, a_0).
for each episode **do**
  if the current s_t is not the target state **then**
    Take action a_t following $\pi_t(s_t)$, generate r_{t+1}, s_{t+1}, and then take action a_{t+1} following $\pi_t(s_{t+1})$
    Value update (parameter update):

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

    Policy update:

$$\pi_{t+1}(a \mid s_t) = \begin{cases} 1 - \frac{\epsilon}{|A(s)|-1} & \text{if } a = \arg\max_a \hat{q}(s_t, a, w_{t+1}) \\ \frac{\epsilon}{|A(s)|} & \text{otherwise.} \end{cases}$$

  end if
end for

Deep Q-learning / Deep Q-network

Q-learning with function approximation]]

Q-learning 只需要将 Sarsa 中的 $\hat{q}(s_{t+1}, a_{t+1}, w_t)$ 替换为 $\max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t)$，就得到

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t) \tag{21}$$

Algorithm 3 Q-learning with Function Approximation (on-policy version)

Initialization: Initial parameter vector w_0. Initial policy π_0. Small $\epsilon > 0$.
Aim: Search a good policy that can lead the agent to the target from an initial state-action pair (s_0, a_0).
for each episode **do**
  if the current s_t is not the target state **then**
    Take action a_t following $\pi_t(s_t)$, and generate r_{t+1}, s_{t+1}.
    Value update (parameter update):

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in A(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

    Policy update: ▷ Since this is on-policy, so exploration needed

$$\pi_{t+1}(a \mid s_t) = \begin{cases} 1 - \frac{\epsilon}{|A(s_t)|-1} & \text{if } a = \arg\max_{a' \in A(s_t)} \hat{q}(s_t, a', w_{t+1}) \\ \frac{\epsilon}{|A(s_t)|} & \text{otherwise.} \end{cases}$$

  end if
end for

Deep Q-learning – basic idea

Deep Q-learning 也被称为Deep Q-network(DQN)，它是最早也是最成功的将深度学习引入强化学习的算法[a]。深度学习在 DQN 中的作用是非线性函数逼近器(nonlinear function approximator)，但是有别于式(21)的是，由于神经网络强大的表达能力，我们不需要再这么底层地来计算 $\hat{q}$.

Objective / Loss function: DQN 目的是最小化如下目标函数/损失函数：

$$J(w) = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)\right)^2\right] \tag{22}$$

其中 (S, A, R, S') 为随机变量.

实际上这里就是在不断逼近 TD target，此 loss function 为 Bellman optimality error，因为根据 BOE 定义：

$$q(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}(S_{t+1})} q(S_{t+1}, a) | S_t = s, A_t = a\right], \quad \forall s, a$$

因此在期望意义下 $R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)$ 应当为0.

Optimize: 最小化损失函数最直接的想法就是使用梯度下降。在损失函数式(22)中参数 w 不仅存在于 $\hat{q}(S, A, w)$ 中，还存在于不好计算梯度的 $\gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w)$ 中，此处定义

$$y \triangleq R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w)$$

将其整体看作 w 是（暂时）固定的，这样就不用计算其梯度.

DQN technique 1: Two networks 在 DQN 中引入了两个网络：

1. **main network:** 第一个网络目的与之前相同，都是为了逼近 $\hat{q}(s, a, w)$
2. **target network:** 第二个网络用于逼近 $\hat{q}(s, a, w_T)$，每隔一段时间就将 w 赋值给 w_T，并持续一段时间内将 $\hat{q}(s, a, w_T)$ 固定、无需进行梯度下降.

这样损失函数就变为了

$$J = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w_T) - \hat{q}(S, A, w)\right)^2\right] \tag{23}$$

这样整个算法和训练的流程就是

将 w 赋值给 $w_T \to$ 固定 w_T 只更新 $\hat{q}(S, A, w)$ 中的 $w \to$ 将 w 赋值给 w_T

DQN technique 2: Experience replay 第二个技巧是使用 experience replay 的方式训练网络：虽然我们收集到数据是有先后顺序的，但是实际使用时，我们需要将其打散，全部归入一个集合 $\mathcal{B} \triangleq \{(s, a, r, s')\}$，称之为 replay buffer，然后使用均匀分布在此集合中采样，就称这些采样为 experience replay，因为有可能被重复取到。这样做的原因在于：根据我们对 loss function 的定义：

$$J = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)\right)^2\right]$$

这其中有四个随机变量 (R, S', S, A)，其中根据 system model 有 $R \sim p(R|S,A), S' \sim p(S'|S,A)$，而我们可以将 (S,A) 看作是一个随机变量[b]，由于赋予高斯分布等需要先验知识（以判断哪些数据是重要的），如若没有则需要一视同仁，即使用均匀分布。由于采集数据时数据是有先后顺序的，因此为实现均匀分布，需要将这些数据打散，破除他们之间的 correlation，这也是使用 replay buffer 的原因.

[a]虽然在 DQN 之前也有一些将深度学习结合强化学习的尝试，但是 DQN 是效果最好的，能够在一些游戏人物上达到人类玩家水准

[b]就像平面坐标下两个坐标表示一个点一样

Questions about DQN

Q: 为什么 tabular Q-learning 不需要 experience replay?
A: 因为在表格形式中根本没有使用到分布的需要，是在不断求解所有 (s,a)的 BOE，而在 DQN 中，损失函数定义使用了期望，因此需要分布.

Q: 可以在表格形式的 Q-learning 中使用 experience replay 吗?
A: 可以！并且这样可以反复使用一次采样的经验，能够让数据(sample)使用地更高效.

DQN Algorithm

Algorithm 4 Deep Q-learning (off-policy version)

1: **Aim:** Learn an optimal target network to approximate the optimal action values from the experience samples generated by a behavior policy π_b.
2: Store the experience samples generated by π_b in a replay buffer $\mathcal{B} = \{(s,a,r,s')\}$.
3: **for** each iteration **do**
4: **Uniformly** draw a mini-batch of samples from $\mathcal{B}$.
5: **for** each sample (s,a,r,s') **do**
6: Calculate the target value as ▷ w_T is the parameter of the target network

$$y_T = r + \gamma \max_{a' \in A(s')} \hat{q}(s', a', w_T),$$

7: **end for**
8: Update the **main network** to minimize

$$(y_T - \hat{q}(s,a,w))^2$$

using the mini-batch $\{(s,a,y_T)\}$.
9: Set $w_T = w$ every C iterations.
10: **end for**

Note 2. 1. 由于算法4是 *off-policy* 的，因此不需要进行 *policy update*

2. 为利用神经网络高效的"黑盒"性，因此不再使用之前基于梯度的 *policy update* 方式

3. *DQN* 原文中的算法是 *on-policy* 的，与此处略有不同

A Stationary Distribution

Introduction

Stationary distribution 描述了马尔可夫决策过程的长期行为，记在 policy π 下马尔可夫过程的 stationary distribution 为 $\{d_\pi(s)\}_{s\in\mathcal{S}}$，即在足够长的时间后 agent 访问 state s 的概率为 $d_\pi(s)$，因此满足

$$\sum_{s\in\mathcal{S}} d_\pi(s) = 1; \quad d_\pi(s) \geqslant 0, \forall s \in \mathcal{S} \tag{24}$$

分析 stationary distribution 的关键工具是概率转移矩阵(probability transition matrix)，记为 $P_\pi \in \mathbb{R}^{n\times n}$. 若 $\mathcal{S} = \{s_1, \cdots, s_n\}$，则 $[P_\pi]_{ij}$ 表示 agent 从 $s_i \to s_j$ 的概率.

Probability transition matrix

记 agent 通过 k 步能够从 state s_i 变为 s_j 的概率为:

$$p_{ij}^{(k)} = \Pr(S_{t_k} = s_j | S_{t_0} = s_i) \tag{25}$$

其中 s_{t_k} 表示第 k 时间步. 那么自然地当 $k = 1$ 时就有

$$[P_\pi]_{ij} = p_{ij}^{(1)} \tag{26}$$

下面再考察 P_π^k. 首先研究 P_π^2:

$$[P_\pi^2]_{ij} = [P_\pi P_\pi]_{ij} = \sum_{q=1}^{n} [P_\pi]_{iq}[P_\pi]_{qj}. \tag{27}$$

由于 $[P_\pi]_{iq}[P_\pi]_{qj}$ 表示 $s_i \to s_q \to s_j$ 的概率，那么 $[P_\pi^2]_{ij}$ 表示使用两步从 $s_i \to s_j$ 的概率. 进一步就可以得到

$$[P_\pi^k]_{ij} = p_{ij}^{(k)}, \tag{28}$$

即 $[P_\pi^k]_{ij}$ 表示使用 k 步从 $s_j \to s_j$ 的概率.

Stationary distribution

记 $d_0 \in \mathbb{R}^n$ 为初始时间步时选择各 state 的概率分布，例如若总选择 s_m 为初始位置则

$$d_0(s_i) = \begin{cases} 1, & i = m \\ 0, & i \neq m. \end{cases} \tag{29}$$

记 $d_k(s_i)$ 为第 k 时间步时选择各 state 的分布，那么就有:

$$\begin{aligned} &\text{elementwise form:} \quad d_k(s_i) = \sum_{j=1}^{n} d_0(s_j)[P_\pi^k]_{ji}, \\ &\text{matrix-vector form:} \quad d_k^T = d_0^T P_\pi^k. \end{aligned} \tag{30}$$

即第 k 步选择 s_i 的概率是通过 k 步由 $\{s_j\}_{j=1}^n$ 变为 s_i 的概率总和.

在特定情形下(详细讨论见)，会满足如下条件

$$\lim_{k\to\infty} P_\pi^k = \mathbf{1}_n d_\pi^T \tag{31}$$

其中 $\mathbf{1}_n = [1,\dots,1]^T \in \mathbb{R}^n$，$\mathbf{1}_n d_\pi^T$ 的每一行都是 d_π^T.

在此条件下，将式(31)代入式(30)中得到:

$$\lim_{k\to\infty} d_k^T = d_0^T \lim_{k\to\infty} P_\pi^k = d_0^T \mathbf{1}_n d_\pi^T \xlongequal{d_0^T \mathbf{1}_n = 1} d_\pi^T \text{.(limiting distribution)} \tag{32}$$

可以看到状态分布 d_k 收敛于常数值 d_π，因此又称 d_π 为 **limiting distribution**，其取决于取决于系统模型与策略 π. 值得注意的是， limiting distribution 独立于 d_0，意味着 agent 从任意 state 开始经过足够长的时间后的概率分布总可以用 limiting distribution 来描述.

How to calculate d_π? 由于 $d_k^T = d_0^T P_\pi^k = d_{k-1}^T P_\pi$，对两边同时取极限，得到

$$d_\pi^T = d_\pi^T P_\pi \text{.(stationary distribution)} \tag{33}$$

因此 d_π 是与特征值1 相关联的 P_π 的左特征向量. 称式(33)的解 d_π 为 **stationary distribution**，其满足 $\sum_{s\in\mathcal{S}} d_\pi(s) = 1; d_\pi(s) > 0, \forall s \in \mathcal{S}$ (严格大于0的原因详见后文).

Note 3. 注意式*(30)*蕴含式*(33)*，但反之不一定成立.

一类具有唯一 limiting / stationary distribution 的马尔可夫过程是不可约(irreducible) / 正则(regular) 马尔可夫过程，与之相关的几个必要概念如下:

1. **Accessible:** $\exists\ 0 < k < \infty, s.t.\ [p_\pi]_{ij}^k > 0$，则称 state s_j 是可从 state s_i 访问的(accessible)，意味着从 s_i 开始的 agent 在有限次数的转换之后有概率到达 s_j.
2. **Irreducible:** $\forall\, i, \forall\, j, \exists\, 0 < k < \infty, s.t.\ [p_\pi]_{ij}^k > 0$，则称之为不可约(irreducible)马尔可夫过程. 意味着所有 state 相互 accessible，从任何 state 开始的 agent 可以在有限步内达到任何其他 state.
3. **Regular:** $\exists\, 0 < k < \infty, s.t.\ \left[P_\pi^k\right]_{ij} > 0, \forall\, i, j$ 或 $\exists\, 0 < k < \infty, s.t.\ P_\pi^k > 0$，则称之为正则(regular)马尔可夫过程. 意味着每个 state 都可以在最多 k 步内从任何 state 到达.

Note 4. 注意 *Regular* 与 *Irreducible* 定义的区别和联系:

1. 正则性定义中 k 是全局性的，意味着 k 确定后所有的 *state* 都可以在 k 步内互相访问，因此可以使用矩阵形式 P_π^k 表示
2. 不可约性定义中 k 是与 i, j 相关的，非全局性，只能表示任意 *state* 间能够互相访问
3. *regular* 蕴含 *irreducible*，但反之未然. 若不可约马尔可夫过程中存在 i 使得 $[P_{ii}] > 0$，那么它也是正则的，因此可以通过不断"原地徘徊"将 k 不断增大

Note 5. 一般来说 *exploratory policy (*如 *ϵ-greedy policy)* 可以生成 *regular Markov processes.*

B Proof

B.1 Proof of Lemma 1

Proof of Lemma 1

Proposition 1 (Lemma 1). 式*(10)*中的期望可以写为

$$\mathbb{E}\left[\left(r_{t+1}+\gamma\phi^T\left(s_{t+1}\right)w_t-\phi^T\left(s_t\right)w_t\right)\phi\left(s_t\right)\right]=b-Aw_t, \tag{34}$$

其中 $A \triangleq \Phi^T D\left(I-\gamma P_\pi\right)\Phi \in \mathbb{R}^{m\times m}$, $b \triangleq \Phi^T D r_\pi \in \mathbb{R}^m$.

Proof. □

B.2 Proof of reversibility

Proof of reversibility

Proposition 2. $A \triangleq \Phi^T D(I-\gamma P_\pi)\Phi \in \mathbb{R}^{m\times m}$ 是可逆*(invertible)*且正定*(positive definite)*的.

Proof. □

B.3 Proof of convergence target under tabular case

Proof of convergence target under tabular case

Proposition 3. 在 *tabular method* 情形下，当 $dim\ w = n = |\mathcal{S}|$，$\phi(s) = [0,\dots,1,\dots,0]^T$时

$$w^* = A^{-1}b = v_\pi.$$

Proof. □

B.4 Proof of equation (16)

Proof of algorithm (16)

Proposition 4.

$$w_t \xrightarrow{t\to\infty} w^* = A^{-1}b.$$

法 *1*：定义收敛误差. □

法 *2*：转为寻根问题. □

B.5 Proof of Theorem1

Proof of Theorem1

Proposition 5. 对于 *projected Bellman error(PBE)* $J_{PBE}(w) = \|\hat{v}(w) - MT_\pi(\hat{v}(w))\|_D^2$ 有

$$w^* = A^{-1}b = \arg\min_w J_{PBE}(w)$$

其中 $M = \Phi(\Phi^T D\Phi)^{-1}\Phi^T D \in \mathbb{R}^{n\times n}$ 为正交投影矩阵.

Proof. □

B.6 Proof of Theorem 2

Proof of Theorem 2

Proposition 6. 在线性情形 $\hat{v}(w^*) = \Phi w^*$ 下，估计值与真实值 v_π 满足

$$\|\hat{v}(w^*) - v_\pi\|_D = \|\Phi w^* - v_\pi\|_D \leq \frac{1}{1-\gamma} \min_w \|\hat{v}(w) - v_\pi\|_D = \frac{1}{1-\gamma} \min_w \sqrt{J_E(w)}. \quad (35)$$

其中 γ 为 *discounted rate.*

Proof. □

First updated: February 14, 2025
Last updated: October 3, 2025

References