2022年11月26日 星期六

强化学习整理

前置概念

- 智能体（三大要素）：
- 感知：智能体在某种程度上感知环境的状态，从而知道自己所处的现状。
- 决策：智能体根据当前的状态计算出达到目标需要采取的动作。
- 奖励：环境根据状态和智能体采取的动作，产生一个标量信号作为奖励反馈。
- 环境：
- 根据当前状态和智能体输入产生下一个状态$\sim P(\dot{E}\,|\,action, state)$
- 目标：将每步的奖励叠加作为回报，我们关注最大化其期望，称为价值。

多臂老虎机（课上没讲但我认为比较重要）

问题形式化表述：

一个元组<A,R>，A代表动作空间$\{a_1, \ldots, a_n\}$,用$a_t \epsilon A$表示任意动作。R代表奖励分布$R(r\,|\,a)$对于任意动作，存在最优动作a*使得奖励的期望最大，我们定义最优奖励为$Q^* = max_{a \epsilon A} Q(a)$，在此基础上我们定义懊悔的概念$R(a) = Q^* - Q(a)$,累计懊悔为对完整的T步决策懊悔的总值$\sigma_R = \sum_{t=1}^{T} R(a_t)$。我们最大化累积奖励，实际上等于最小化累积期望。

估计期望奖励

- 对于$\forall a \in \mathcal{A}$，初始化计数器 $N(a) = 0$和期望奖励估值 $\hat{Q}(a) = 0$
- **for** $t = 1 \rightarrow T$ **do**
- 选取某根拉杆，该动作记为 a_t
- 得到奖励 r_t
- 更新计数器: $N(a_t) = N(a_t) + 1$
- 更新期望奖励估值:

$$\hat{Q}(a_t) = \hat{Q}(a_t) + \frac{1}{N(a_t)}\left[r_t - \hat{Q}(a_t)\right]$$

- **end for**

增量式方法：

$$
\begin{aligned}
Q_k &= \frac{1}{k}\sum_{i=1}^{k} r_i \\
&= \frac{1}{k}\left(r_k + \sum_{i=1}^{k-1} r_i\right) \\
&= \frac{1}{k}(r_k + (k-1)Q_{k-1}) \\
&= \frac{1}{k}(r_k + kQ_{k-1} - Q_{k-1}) \\
&= Q_{k-1} + \frac{1}{k}[r_k - Q_{k-1}]
\end{aligned}
$$

ϵ-贪心算法：

每次采取的动作不是当前估计的Q*，而是以一定概率随机选择：

$$
a*t = \begin{cases} \arg\max *a \in \mathcal{A}\hat{Q}(a), & \text{采样概率：1-}\epsilon \\ \text{从 } \mathcal{A} \text{ 中随机选择，} & \text{采样概率：}\epsilon \end{cases}
$$

引入ϵ-贪心算法的原因是平衡探索（探索新摇杆的期望）和利用（选择已知期望的最优值）

马尔可夫过程

马尔可夫决策过程

随机过程：随时间变化的随机现象（可以理解为分布的参数是时间的函数）

马尔可夫性质：当且仅当某时刻的状态只取决于上一时刻的状态

$P(S_{t+1}|S_T) = P(S_{t+1}|S_1, \ldots, S_t)$

马尔可夫过程：具有马尔可夫性质的随机过程，可以通过状态转移矩阵表示

$$
H = \begin{bmatrix} P(s_1|s_1) & \cdots & P(s_n|s_2) \\ \vdots & \ddots & \vdots \\ P(s_1|s_n) & \cdots & P(s_n|s_n) \end{bmatrix}
$$

马尔可夫奖励过程

在马尔可夫过程的基础上加入奖励函数 和折扣因子，就可以得到马尔可夫奖励过程

由<S,P,r,γ>构成。r是奖励函数，代表从某个状态转移到特定s状态的奖励。γ是折扣因子。

回报：

在一个马尔可夫奖励过程中，从第t时刻状态S_t开始，到终止状态时，所有奖励的衰减之和称为回报$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$

价值函数：

一个状态的期望回报（从这个状态出发的未来累计奖励的期望）成为状态价值

$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | S_t = s] \\
&= \mathbb{E}[R_t + \gamma(R_{t+1} + \gamma R_{t+2} + \ldots) | S_t = s] \\
&= \mathbb{E}[R_t + \gamma G_{t+1} | S_t = s] \\
&= \mathbb{E}[R_t + \gamma V(S_{t+1}) | S_t = s]
\end{aligned}
$$

同时，状态价值可以表示为$V(s) = r(s) + \gamma \sum_{s' \epsilon S} p(s'|s) V(s')$

写成矩阵形式，我们得到了**Bellman Equation**：$V = R + \gamma P V$

通过求解矩阵，我们得到状态价值函数的解析解：$V = (I - \gamma P)^{-1} R$

但改矩阵的计算复杂度过大，因此后续引入了动态规划算法，蒙特卡洛方法和时序差分算法进行估计。

马尔可夫决策过程：

马尔可夫决策元组由<S,A,P,r,γ>构成，相较之前，变化在于r(s,a)表示在s状态采取a动作的奖励，$P(s'|s,a)$表示在s状态下采取a动作转移到s'状态的概率。

策略（policy）

$\pi(a|s) = P(A_t = a|S_t = s)$表示在输入st状态情况下采取动作at的概率

在策略的基础上，我们可以得到不同策略的状态价值函数，定义为从状态s出发遵循策略π能获得的期望回报:$V^{\pi}(s) = E_{\pi}[G_t|S_t = s]$

接着，有策略价值函数（代表在当前状态st采取策略at的期望回报）

$Q^{\pi}(s,a) = E_{\pi}[G_t|S_t = s, A_t = a]$

同时，根据策略的定义，我们有如下两个等式

$$V^{\pi}(s) = \sum_{a \epsilon A} \pi(a|s) Q^{\pi}(s,a)$$

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s' \epsilon S} P(s'|s,a) V^{\pi}(s')$$

代入联立得到Bellman Exception equation

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[R_t + \gamma V^{\pi}(S_{t+1})|S_t = s] \\ &= \sum_{a \in A} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi}(s') \right) \\ Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[R_t + \gamma Q^{\pi}(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \\ &= r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q^{\pi}(s', a') \end{aligned}$$

如果我们定义奖励$r'(s) = \sum_{a \epsilon A} \pi(a|s) r(s,a)$,状态转移函数

$P'(s'|s) = \sum_{a \epsilon A} \pi(a|s) P(s'|s,a)$，则可将马尔可夫决策过程转换为马尔可夫奖励过程，可以利用Bellman equation：$V = (I - \gamma P)^{-1} R$ 解析计算其状态价值函数V(s)。

贝尔曼最优方程

首先定义策略之间的偏序关系：当且仅当对于任意的状态s都有$V^{\pi}(s) \geq V^{\pi'}(s)$,记$\pi > \pi'$.于是在有限状态和动作集合里，必有一个策略不差于其他所有策略，这个策略就是最优策略（optimal policy）对应的价值函数称为最优价值函数和最优价值函数：

$$V^*(s) = \underbrace{max}_{\pi} V^{\pi}(s) \qquad Q^*(s,a) = \underbrace{max}_{\pi} Q^* \pi(s,a)$$

为了使$Q^{\pi}(s,a)$最大，我们需要在当前的状态动作对(s,a)之后都执行最优策略。二者之间的关系为：

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \epsilon S} P(s'|s,a) V^*(s')$$

$$V^{\pi}(s) = \underbrace{max}_{a \epsilon A} Q^*(s,a)$$

最终联立我们得到贝尔曼最优方程：

$$V^*(s) = \max_{a \in \mathcal{A}} \{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \}$$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}} Q^*(s',a')$$

动态规划算法

核心思想：

动态规划的基本思想是将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到目标问题的解。动态规划会保存已解决的子问题的答案，在求解目标问题的过程中，需要这些子问题答案时就可以直接利用，避免重复计算。

类别：

策略迭代：由策略评估和策略提升两步构成

价值迭代

策略迭代算法：

策略评估：回顾贝尔曼期望方程

$$V^{\pi}(s)=\sum_{a\epsilon A}\pi(a|s)(r(s,a)+\gamma\sum_{s'\epsilon S}p(s'|s,a)V^{\pi}(s')$$

可以看到，当知道奖励函数和状态转移函数时，我们可以根据下一个状态的价值来计算当前状态的价值。

因此，可以根据动态规划的思想，把计算下一个可能状态的价值当成一个子问题，把计算当前状态的价值看作当前问题。在得知子问题的解后，就可以求解当前问题。更一般的，在考虑所有状态的情况下，可以用上一轮的状态价值函数估计这一轮的状态价值函数。

$$V^{k+1}(s)=\sum_{a\in A}\pi(a|s)\left(r(s,a)+\gamma\sum_{s'\in S}P(s'|s,a)V^{k}(s')\right)$$

可以证明，$V^{k}=V^{\pi}$是上述方程的不动点，在$k\rightarrow\infty$时，序列V^{k}会收敛到V^{π}

策略提升：

当我们使用策略评估计算得到当前策略的状态价值函数之后，我们可以据此来改进该策略。具体来说可以直接贪心地在每一个状态选择动作价值最大的动作，也就是

$$\pi'(s)=argmax_{a}Q^{\pi}(s,a)=argmax_{a}\{r(s,a)+\gamma\sum_{s'}P(s'|s,a)V^{\pi}(s')\}$$

当策略提升之后得到的策略π'和之前的策略π一样时，说明策略迭代达到了收敛，此时π和π'就是最优策略。

- 随机初始化策略 $\pi(s)$ 和价值函数 $V(s)$
- **while** $\Delta > \theta$ **do**：（策略评估循环）
- $\Delta \leftarrow 0$
- 对于每一个状态 $s \in \mathcal{S}$:
- $v \leftarrow V(S)$
- $V(s) \leftarrow r(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V(s')$
- $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
- **end while**
- $\pi_{\text{old}} \leftarrow \pi$
- 对于每一个状态 $s \in \mathcal{S}$:
- $\pi(s) \leftarrow \arg\max_a r(s, a) + \gamma \sum_{s'} P(s'|s, a) V(s')$
- 若$\pi_{\text{old}} = \pi$, 则停止算法并返回 V 和 π; 否则转到策略评估循环

价值迭代算法：

策略迭代算法中需要多次进行策略评估和策略提升才能收敛带来很大的计算量，但是存在状态价值函数还没有收敛，但是不论接下来怎么更新状态价值，策略提升得到的都是同一个策略。因此可以只在策略评估中进行一轮价值更新，然后直接根据更新后的价值进行策略提升。这就是价值迭代算法，它可以被认为是一种策略评估只进行了一轮更新的策略迭代算法。价值迭代中不存在显式的策略，我们只维护一个状态价值函数。

回顾贝尔曼最优方程：

$$V^*(s) = max_{a \epsilon A}\{r(s,a) + \gamma \sum_{a' \epsilon S} P(s'|s,a) V^*(s')\}$$

改写为迭代更新形式：

$$V^{k+1}(s) = max_{a\epsilon A}\{r(s,a) + \gamma \sum_{a'\epsilon S} P(s'|s,a)V^k(s')\}$$

等到V^{k+1}和V^k相同时，它就是贝尔曼最优方程的不动点。此时直接利用$\pi(s) = argmax_a\{r(s,a) + \gamma \sum_{s'} P(s'|s,a)V^{k+1}(s')\}$得出最优策略即可。

伪代码如下：

- 随机初始化$V(s)$
- **while** $\Delta > \theta$ **do**:
- $\Delta \leftarrow 0$
- 对于每一个状态 $s \in \mathcal{S}$:
- $v \leftarrow V(s)$
- $V(s) \leftarrow \max_a r(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')$
- $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
- **end while**
- 返回一个确定性策略
  $\pi(s) = \arg\max_a\{r(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')\}$

蒙特卡洛方法

蒙特卡洛方法的主要应用是在时序差分算法里，这里简单总结一下蒙特卡洛方法的采样估计过程，时序差分算法部分会详细展开。

对于马尔可夫决策过程，一个状态的价值是它的期望回报，那么一个很简单直接的想法便是用策略在 MDP 上采样很多条序列，计算从这个状态出发的回报再求其期望就可以了$V^{\pi}(s) = E_{\pi}[G_t|S_t = s] \approx \frac{1}{N}\sum_{i=1}^{N} G_t^{(i)}$

在一条序列中，可能没有出现过这个状态，可能只出现过一次这个状态，也可能出现过很多次这个状态。因此存在两种计算蒙特卡洛价值估计的方法，一种是这条序列第一次出现该状态时计算后面的累积奖励，而后面再次出现该状态时，该状态就被忽略了。一种是在该状态每一次出现时计算它的回报。这里只介绍较为复杂的第二种方法，第一种方法大同小异。

计算过程如下：

（1）使用策略π采样若干条序列：

$$s_0^{(i)} \xrightarrow{a_0^{(i)}} r_0^{(i)}, s_1^{(i)} \xrightarrow{a_1^{(i)}} r_1^{(i)}, s_2^{(i)} \xrightarrow{a_2^{(i)}} \cdots \xrightarrow{a_{T-1}^{(i)}} r_{T-1}^{(i)}, s_T^{(i)}$$

（2）对于每条序列：

更新状态s的计数器$N(s) \leftarrow N(s)+1$

更新状态s的总回报$M(s) \leftarrow M(s)+G_i$

（3）每一个状态的价值被估计为回报的平均值 $V(s)=M(s)/N(s)$

同时，可以使用增量式更新的方式计算：

$$N(s) \leftarrow N(s)+1$$

$$V(s) \leftarrow V(s)+\frac{1}{N(s)}(G-V(S))$$

离线蒙特卡洛采样：

有时用于产生序列的策略和我们所需要估计的策略是不一样的，离线的具体训练方法中后面离线Q-learning中会具体介绍，这里只讲一下大体思路

定义占用度量：

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1}\pi(A_k|S_k)p(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{T-1}b(A_k|S_k)p(S_{k+1}|S_k,A_k)} = \prod_{k=t}^{T-1}\frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

该式计算的是采取目标策略和采样策略生成同样的序列概率的比值，之后在利用蒙特卡洛法估计目标策略的价值函数时只需要在序列的每个状态的回报乘上该因子的归一化形式再用这条序列的回报估计目标策略即可。

其余的对于动作价值函数的蒙特卡洛估计会在时序差分算法里涉及

时序差分算法

简介

动态规划算法要求与智能体交互的环境是完全已知的。在此条件下，智能体其实并不需要和环境真正交互来采样数据，直接用动态规划算法就可以解出最优价值或策略。

而真实的强化学习场景中，其马尔可夫决策过程的状态转移概率是无法写出来的，也就无法直接进行动态规划。。在这种情况下，智能体只能和环境进行交互，通过采样到的数据来学习，这类学习方法统称为无模型的强化学习。

不同于动态规划算法，无模型的强化学习算法不需要事先知道环境的奖励函数和状态转移函数，而是直接使用和环境交互的过程中采样到的数据来学习。而时序差分算法就是一类无模型的强化学习方法。

时序差分

回顾蒙特卡洛方法对价值函数的增量更新方式$V(s_t) \leftarrow V(s_t) + \alpha[G_t - V(s_t))]$

为了得到回报G_t，蒙特卡洛方法必须要等整个序列结束之后才能计算得到这一次的回报，而时序差分方法只需要当前步结束即可进行计算。具体来说，时序差分算法用当前获得的奖励加上下一个状态的价值估计来作为在当前状态会获得的回报，即：

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

其中$R_t + \gamma V(s_{t+1}) - V(s_t)$被称为时序差分,$\alpha$为更新步长。

Sarsa算法

我们可以用时序差分方法来估计价值函数，也可以直接用时序差分算法来估计动作价值函数：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

我们每次选取动作都选取$a_t = argmax_a Q(s, a)$,用贪婪算法根据动作价值选取动作来和环境交互，再根据得到的数据用时序差分算法更新动作价值估计。

这样可能导致某些状态动作对(s,a)无法对其动作价值进行估计而直接选择确定性策略

故采取ϵ贪心策略：

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{|A|} + 1 - \epsilon, & a = argmax_a' Q(s, a') \\ \frac{\epsilon}{|A|}, & others \end{cases}$$

完整流程的伪代码如下：

- 初始化$Q(s, a)$
- **for** 序列$e = 1 \rightarrow E$ **do**:
- 　　得到初始状态s
- 　　用ϵ-greedy 策略根据Q选择当前状态s下的动作a
- 　　**for** 时间步$t = 1 \rightarrow T$ **do**:
- 　　　　得到环境反馈的r, s'
- 　　　　用ϵ-greedy 策略根据Q选择当前状态s'下的动作a'
- 　　　　$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
- 　　　　$s \leftarrow s', a \leftarrow a'$
- 　　**end for**
- **end for**

Q-learning

Q-learning 和 Sarsa 的最大区别在于 Q-learning 的时序差分更新方式为

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

伪代码为：

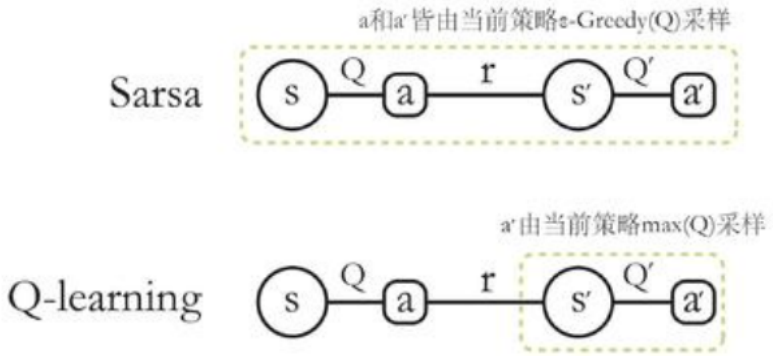
- 初始化$Q(s,a)$
- **for** 序列$e=1\rightarrow E$ **do**:
- 　　得到初始状态s
- 　　**for** 时间步$t=1\rightarrow T$ **do**:
- 　　　　用ϵ-greedy 策略根据Q选择当前状态s下的动作a
- 　　　　得到环境反馈的r,s'
- 　　　　$Q(s,a)\leftarrow Q(s,a)+\alpha[r+\gamma \max_{a'}Q(s',a')-Q(s,a)]$
- 　　　　$s\leftarrow s'$
- 　　**end for**
- **end for**

我们可以用价值迭代的思想来理解Q-learning，即Q-learning是直接在估计Q*，因为动作价值函数的贝尔曼最优方程是

$$Q^*(s,a)=r(s,a)+\gamma\sum_{s'\epsilon S}P(s'|s,a)max_{a'}Q^*(s',a')$$

而Sarsa估计的是ϵ-贪婪策略的价值函数。这也导致Q-learning的结果一般比Sarsa算法要好。

同时可以注意到，Q-learning 的更新并非必须使用当前贪心策略$argmax_aQ(s,a)$得到的数据，因为给定任意(s,a,r,s')都可以直接根据更新公式来更新Q。Sarsa 必须使用当前ϵ-贪婪策略采样得到的数据，因为它的更新中用到的Q(s',a')的a'是当前策略在s'下的动作。因此，Sarsa是在线策略，Q-learning是离线策略。



多步Sarsa算法

时序差分方法的缺点：

时序差分算法具有非常小的方差，因为只关注了一步状态转移，用到了一步的奖励，但是它是**有偏**的，因为用到了下一个状态的价值估计而不是其真实的价值。蒙特卡洛方法是**无偏**的，但是具有比较大的方差，因为每一步的状态转移都有不确定性，而每一步状态采取的动作所得到的不一样的奖励最终都会加起来，这会极大影响最终的价值估计。

两者结合：多步时序差分

使用n步的奖励，然后使用之后状态的价值估计。

具体来说,，将$G_t = r_t + \gamma Q(s_{t+1}, a_{t+1})$

替换成：$G_t = r_t + \gamma r_{t+1} + \ldots + \gamma^n Q(s_{t+n}, a_{t+n})$

故多步Sarsa将

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

替换成：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma r_{t+1} + \cdots + \gamma^n Q(s_{t+n}, a_{t+n}) - Q(s_t, a_t)]$$

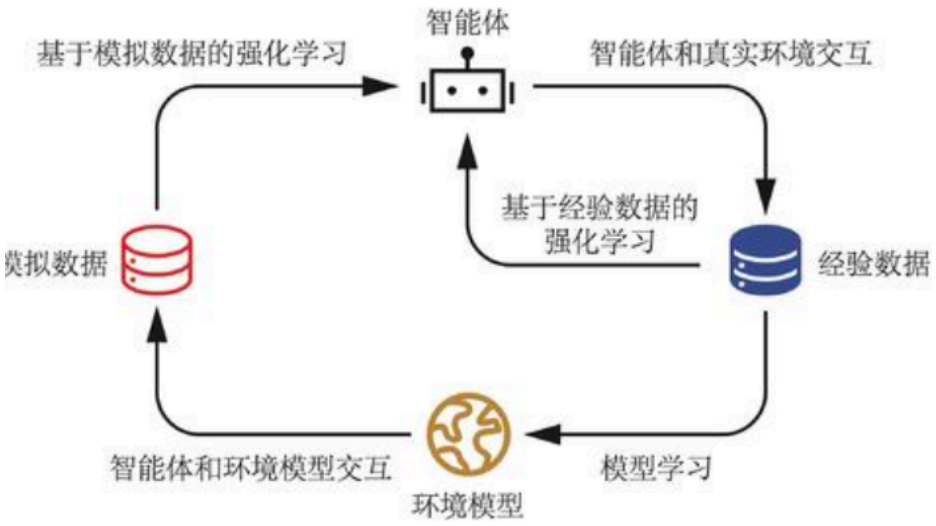
Dyna-Q算法（基于模型的强化学习）

模型

在强化学习中，“模型”通常指与智能体交互的环境模型，即对环境的状态转移概率和奖励函数进行建模。根据是否具有环境模型，强化学习算法分为两种：基于模型的强化学习和无模型的强化学习。无模型的强化学习根据智能体与环境交互采样到的数据直接进行策略提升或者价值估计(Sarsa and Q-learning)，在基于模型的强化学习中，模型可以是事先知道的，也可以是根据智能体与环境交互采样到的数据学习得到的，然后用这个模型帮助策略提升或者价值估计(Dynamic Programming)。Dyna-Q算法也是基于模型的强化学习算法，不过它的环境模型是通过采样数据估计得到的。

基于模型的强化学习算法由于具有一个环境模型，智能体可以额外和环境模型进行交互，对真实环境中样本的需求量往往就会减少，因此通常会比无模型的强化学习算法具有更低的样本复杂度。但是，环境模型可能并不准确，不能完全代替真实环境，因此基于模型的强化学习算法收敛后其策略的期望回报可能不如无模型的强化学习算法。

Dyna-Q



具体流程为：

初始化$Q(s,a)$，初始化模型$M(s,a)$

for 序列$e=1\rightarrow E$ **do**:

得到初始状态s

for $t=1\rightarrow T$ **do**:

用ϵ-贪婪策略根据Q选择当前状态s下的动作a

得到环境反馈的r,s'

$Q(s,a)\leftarrow Q(s,a)+\alpha[r+\gamma\max_{a'}Q(s',a')-Q(s,a)]$

$M(s,a)\leftarrow r,s'$

for 次数 $n=1\rightarrow N$ **do**:

随机选择一个曾经访问过的状态s_m

采取一个曾经在状态s_m下执行过的动作a_m

$r_m,s'_m\leftarrow M(s_m,a_m)$

$Q(s_m,a_m)\leftarrow Q(s_m,a_m)+\alpha[r_m+\gamma\max_{a'}Q(s'_m,a')-Q(s_m,a_m)]$

end for

$s\leftarrow s'$

end for

end for