强化学习基础

Jianjun Wu

December 21, 2019

1 基本概念

强化学习中最基本的MDP(Markov decision process)如下:

Algorithm 1 强化学习过程

```
t = 1;
while True do
    agent观察到系统的状态s_t;
    agent根据策略π(s_t)执行动作a_t;
    系统的状态以概率p(s_{t+1}|s_t, a_t)转移到s_{t+1};
    agent收到回报r_t;
    t = t + 1;
end while
```

强化学习的目标就是最大化累计回报R，策略π的累计回报R^π有两种定义：

$$\begin{aligned} R^\pi &= \lim_{T\to\infty} \frac{\mathbb{E}\left(\sum_{t=1}^{T} r_t|\pi\right)}{T}, && \text{平均回报} \\ R^\pi &= \lim_{T\to\infty} \mathbb{E}\left(\sum_{t=1}^{T} \gamma^{t-1} r_t|\pi\right), && \text{折扣回报,} 0<\gamma<1 \end{aligned} \tag{1}$$

我们定义MDP中整个状态空间为$\mathcal{S}$，整个动作空间为$\mathcal{A}$。强化学习有三大组件：

1. policy(策略)：分确定性策略$\pi(s)$(状态s下应该采取哪个动作)和随机性策略两种$\pi(a|s)$(状态s下采取每个动作的概率)。

2. value function(值函数)：表示了agent根据策略π做动作时，在某个系统状态s下对的接下来累计回报的估计：

   $$V^\pi(s) = \mathbb{E}(R^\pi|s,\pi) \tag{2}$$

   , 或者agent在某个系统状态s下采取某个动作a后对接下来累计回报的估计：

   $$Q^\pi(s,a) = \mathbb{E}(R^\pi|s,a,\pi) \tag{3}$$

   我们还常常使用优势函数：

   $$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{4}$$

3. model(模型)：包括两个矩阵，一个是在状态s下采取动作a 以后，系统转移到状态s'的概率$P(s'|s,a)$。另一个是，在状态s下采取动作a所获得的回报的期望值$R(s,a) = \mathbb{E}(r|s,a)$，注意即便$s,a$确定后，其回报也是随机的，所以这里说的是期望。

对于折扣回报，可以得到下面的结果：

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) \sum_{a'\in\mathcal{A}} \pi(a'|s') Q^\pi(s',a') \tag{5a}$$

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) V^\pi(s') \tag{5b}$$

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) V^\pi(s') \tag{5c}$$

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) Q^\pi(s,a) \tag{5d}$$

我们定义策略间的偏序关系：

$$\pi \geq \pi' \quad \text{if} \quad V^{\pi}(s) \geq V^{\pi'}(s), \forall s \in \mathcal{S}$$

我们称策略π^*是最优策略当且仅当:$\pi^* \geq \pi, \text{for} \, \forall \pi$。最优策略有几条重要性质：

$$V^{\pi_*}(s) = \max_a Q^{\pi_*}(s,a) \tag{6a}$$

$$V^{\pi_*}(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{\pi_*}(s') \right) \tag{6b}$$

$$Q^{\pi_*}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a'} Q^{\pi_*}(s',a') \tag{6c}$$

方程(6)又称Bellman方程。

MDP一定存在最优策略，MDP就是要求解最优策略。在知道$Q^{\pi_*}(s,a)$后，最优策略如下：

$$\pi_*(a|s) = \begin{cases} 1, & \text{if } a = \underset{a' \in \mathcal{A}}{\text{argmax}} Q^{\pi_*}(s,a') \\ 0, & \text{否则} \end{cases} \tag{7}$$

MDP有很多扩展，比如：

- 状态扩展：无限的状态空间或者状态空间是连续的。
- 动作扩展：无限的动作空间或者动作空间是连续的。
- POMDP：状态不可直接观测。

有很多算法用于求解MDP的最优策略，接下来我们就一一介绍。

2 查表算法

所谓查表算法，也就是我们需要计算出每个$Q(s,a), \forall (s,a) \in \mathcal{S} \times \mathcal{A}$,或者每个$V(s), \forall s \in \mathcal{S}$，然后根据这些值生成最优策略。我们这里考虑确定性策略$\pi(s)$。对于确定策略而言，公式(5)可以简化为：

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) Q^{\pi}(s', \pi(s')) \tag{8a}$$

$$V^{\pi}(s) = R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,\pi(s)) V^{\pi}(s') \tag{8b}$$

$$V^{\pi}(s) = Q^{\pi}(s,\pi(s)) \tag{8c}$$

首先看看基于动态规划的策略迭代算法(算法2)和值迭代算法(算法3)。需要指出的是，虽然这里我们给出的策略迭代算法和值迭代算法是基于Q值的，其实也可以基于V值。可根据公式(8b)和(6b)分别导出基于V值得策略迭代和值迭代算法，可由公式(5b)导出所有的Q值。

可以证明算法2和算法3是收敛的，并且收敛到最优策略。可以看到算法2和算法3都依赖转移矩阵$p(s'|s,a)$ 和回报矩阵$R(s,a)$，也就是所谓的model，然而在很多情况下，我们却不知道这两个信息，此时我们需要model-free的算法。

Algorithm 2 基于Q值的策略迭代

1: $k=0$，初始化策略π_0。
2: **while** True **do**
3: 评价当前策略π_k：初始化策略π_k的Q值：$\forall(s,a)\in\mathcal{S}\times\mathcal{A}, Q_k^{\pi}(s,a)=0$。然后根据下面的公式(其实就是公式8a)迭代计算$Q^{\pi_k}$直到收敛。

$$Q^{\pi_k}(s,a)=R(s,a)+\gamma\sum_{s'\in\mathcal{S}}p(s'|s,a)Q^{\pi_k}(s,\pi_k(s')')$$

4: 改进当前策略：$\pi_{k+1}(s)=\underset{a'\in\mathcal{A}}{\text{argmax}}Q^{\pi_k}(s,a')$。
5: 如果$\pi_k=\pi_{k+1}$则$\pi^*=\pi_{k+1}$，算法停止，否则$k=k+1$;
6: **end while**

Algorithm 3 基于Q值的值迭代

1: 初始化Q值：$\forall(s,a)\in\mathcal{S}\times\mathcal{A}, Q_0(s,a)=0$，$k=0$。
2: **while** True **do**
3: 根据下面的公式计算Q_{k+1}(其实就是公式6c)。

$$Q_{k+1}(s,a)=R(s,a)+\gamma\sum_{s'\in\mathcal{S}}p(s'|s,a)\max_{a'}Q_k(s',a')$$

4: 计算策略

$$\pi_k(s)=\underset{a'\in\mathcal{A}}{\text{argmax}}Q_k(s,a'),\pi_{k+1}(s)=\underset{a'\in\mathcal{A}}{\text{argmax}}Q_{k+1}(s,a')$$

5: 如果$\pi_k=\pi_{k+1}$则跳出循环，否则$k=k+1$;
6: **end while**
7: 最优策略为:$\pi^*(s)=\underset{a'\in\mathcal{A}}{\text{argmax}}Q_{k+1}(s,a')$。

有很多model-free算法如，TD(Temporal-Difference)，MC(Monte-Carlo)等。他们都有一个共同特点，那就是agent不断地探索系统，跟系统互动，得到许多轨迹样本$\{(s_t,a_t,r_t)\},t=1,2,\ldots,\infty$，然后根据这些轨迹样本估算Q和V值。为了尽可能探索到整个空间，通常采用ϵ-greedy策略，如下($m=|\mathcal{A}|$):

$$\pi(a|s)=\begin{cases}\frac{\epsilon}{m}+1-\epsilon & \text{if } a=\underset{a'\in\mathcal{A}}{\text{argmax}}Q(s,a')\\ \frac{\epsilon}{m} & \text{否则}\end{cases}\tag{9}$$

我们先介绍一个浅显但是却很有用的统计学技巧。

Proposition 2.1. 概率期望

$$Y=\mathbb{E}_{X\sim p(X)}\left[f(X)\right]=\int p(x)f(x)dx$$

可以通过采样估计，从概率分布$p(X)$ 采样得n个样本：$x_i,i=1,2,\ldots,n$，那么

$$\frac{1}{n}\sum_{i=1}^{n}f(x_i)$$

是Y的一个无偏估计。

命题2.1说明，如果一个值是某个概率分布的期望，那么我们就可以通过采样的方式来估计它，而我们的Q值和V值正是一种概率期望值，所以我们也可以通过采样来估计。

$$\begin{aligned} Q^{\pi}(s_t, a_t) &= \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | \, \pi\right] \\ &= \mathbb{E}\left[r_t\right] + \gamma \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \cdots | \, \pi\right] \\ &= \mathbb{E}\left[r_t\right] + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) \end{aligned} \tag{10}$$

对于最优策略则有：

$$\begin{aligned} Q^{\pi_*}(s_t, a_t) &= R(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t, a_t) \max_{a'} Q^{\pi_*}(s_{t+1}, a') \\ &= \mathbb{E}(r|s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p(s_{t+1}|s,a)} \left[\max_{a'} Q^{\pi_*}(s_{t+1}, a')\right] \\ &= \mathbb{E}(r_t) + \gamma \mathbb{E}_{s_{t+1} \sim p(s_{t+1}|s,a)} \left[\max_{a'} Q^{\pi_*}(s_{t+1}, a')\right] \end{aligned} \tag{11}$$

所以$r_t + \gamma \max_{a'} Q^{\pi_*}(s_{t+1}, a')$ 是$Q^{\pi_*}(s_t, a_t)$ 的一个无偏估计，$r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})$是$Q^{\pi}(s_t, a_t)$的一个无偏估计。

Sarsa算法(4)用于策略评价，也就是计算某个策略的Q值。在Sarsa算法中，agent不断地采取动作探索系统，系统状态不断地转移，直到系统转移到一个终结状态，则完成一个episode，然后系统被随机地初始化，继续下一轮探索。

Algorithm 4 Sarsa算法

初始化Q值：$\forall (s, a) \in \mathcal{S} \times \mathcal{A}, Q(s, a) = 0$。
for 每一个episode **do**
    初始化系统状态$s_0, t = 0$。
    while s_t不是终结状态 **do**
        根据ϵ-greedy策略选择在状态s_t下的一个动作a_t并执行。
        系统反馈回报r_t并转移到新状态s_{t+1}，再根据ϵ-greedy策略选择在状态s_{t+1}下的一个动作a_{t+1}。
        更新Q值：$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
        $t = t + 1$。
    end while
end for

Q-learning算法 [1]跟Sarsa算法很相似，计算过程如**Algorithm**5。

Algorithm 5 Q-learning算法

初始化Q值：$\forall (s, a) \in \mathcal{S} \times \mathcal{A}, Q(s, a) = 0$。
for 每一个episode **do**
    初始化系统状态$s_0, t = 0$。
    while s_t不是终结状态 **do**
        根据ϵ-greedy策略选择在状态s_t下的一个动作a_t并执行。
        系统反馈回报r_t并转移到新状态s_{t+1}。
        更新Q值：$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$
        $t = t + 1$。
    end while
end for

3 值近似算法

查表算法需要存储每个(状态，动作)对应的Q值或者每个状态对应的V 值，这对于状态空间巨大或者状态空间连续的场合是不适用的。我们可采用带参函数近似的方法来获得每个Q 值或者每个V值。

$$Q(s,a;\mathbf{w}) \approx Q(s,a), V(s;\mathbf{w}) \approx V(s)$$

我们介绍颇具有代表性的一个值近似算法：DQN [2]。它利用神经网络拟合Q函数，输入是视频中连续4帧图片，表示系统状态，输出则是在当前状态下采取每个动作的期望回报值，如此给出了每个(s,a)的Q值估计。它有两个重要特点：

- DQN中有两个网络Q和$\hat{Q}$，他们的网络结构完全相同，初始参数也相同。先固定网络$\hat{Q}$，而训练网络Q。每隔一段时间把网络Q中的被训练更新后的参数赋值给网络$\hat{Q}$。
- 利用经验回放实现训练样本的iid，也就是从初始状态开始，不断采取随机动作导致系统状态转移，从而形成大量转移样本，然后随机采样一部分转移样本作为神经网络的训练样本。

DQN算法计算过程如**Algorithm**6。

Algorithm 6 带经验回放的DQN算法.

1: 初始化回放空间D为空;
2: 用随机初始化的参数θ初始化网络Q;
3: 用参数$\hat{\theta}=\theta$初始化网络$\hat{Q}$;
4: **for** 每个episode **do**
5: 设系统当前图片为x_1,取其相邻的4帧图片形成s_1
6: **for** $t \in [1,T]$ **do**
7: 根据网络$\hat{Q}$对每个(s,a)的值估计，采用ϵ-greedy策略选择在状态s_t下的一个动作a_t;
8: 在模拟器中执行动作a_t,并得到回报r_t以及系统转移后的新状态x_{t+1}，取x_{t+1}相邻的4帧图片形成s_{t+1};
9: 把转移样本$d_t=(s_t,a_t,r_t,s_{t+1})$ 存入回放空间D;
10: 从回放空间D中采样一批转移样本D_t;设$D_t(a)$表示转移样本集合D_t中所有动作的集合;
11: 令$error=0$;
12: **for** $d_j=(s_j,a_j,r_j,s_{j+1}) \in D_t$ **do**
13: $$\hat{y}_j=\begin{cases} r_j, & \text{如果}s_{j+1}\text{是终结状态} \\ r_j+\gamma \max\limits_{a' \in D_t(a)} \hat{Q}(s_{j+1},a'), & \text{否则} \end{cases}$$
14: $error=error+(y_j-Q(s_j,a_j))^2$。
15: **end for**
16: 利用误差$error$更新网络Q的参数θ。
17: 每隔C步便用网络Q的参数θ更新网络$\hat{Q}$的参数$\hat{\theta}$，$\hat{\theta}=\theta$。
18: **end for**
19: **end for**

DQN有很多改进：Double DQN [3]改进了a'的选择(13行)；Prioritized Replay DQN [4]根据转移样本的TD(0)误差进行优先级采样；Duelling network [5]将整

个Q值网络分为两个部分，一个用于计算状态的值函数$V^\pi(s)$，另一个用于计算优势函数$A^\pi(s,a)$。Gorila [6]是一种分布式训练DQN的平台。

4 策略梯度算法

因为DQN算法输出的是一个$|\mathcal{A}|$维的向量，表示在输入状态下每个动作的回报值。可以看出当动作空间很大或者连续时，DQN算法将失效。此时，策略梯度算法将派上用场。策略梯度算法就是用一个带参函数$\pi = \pi(a|s;\theta)$ 表示策略，然后迭代地更新参数θ使得某个策略目标函数$J(\pi;\theta)$最优。

REINFORCE算法 [7]是一种比较早的策略梯度算法，计算过程如**Algorithm7**：

Algorithm 7 REINFORCE算法

1: 初始化参数θ_1。
2: **for** $k \in [1, K]$ **do**
3: **for** $i \in [1, N]$ **do**
4: 从系统初始状态开始，运行策略$\pi(a|s;\theta)$直到遇到终结状态，得到第i条轨迹 $\tau^i = (s_0^i, a_0^i, r_0^i), (s_1^i, a_1^i, r_1^i), \ldots, (s_T^i, a_T^i, r_T^i)$。
5: **end for**
6: 根据N个轨迹样本更新θ值：$\theta_{k+1} = \theta_k + \alpha \sum_i (\sum_t \nabla_\theta \log \pi(a_t^i|s_t^i;\theta))(\sum_t r_t^i)$。
7: **end for**

REINFORCE算法优化的目标函数其实为:

$$J(\pi;\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)} R(\tau)$$

其中$\pi_\theta(\tau)$表示在策略π下轨迹τ的概率，$R(\tau)$表示轨迹τ上的所有回报之和。而对于我们通常的折扣回报，策略目标函数为：

$$\begin{aligned}
J(\pi;\theta) &= \lim_{T\to\infty} \mathbb{E}\left(\sum_{t=1}^T \gamma^{t-1} r_t|\pi\right) \\
&= \lim_{T\to\infty} \left(\sum_{t=1}^T \gamma^{t-1} \sum_{s\in\mathcal{S}} p(s_t = s|\pi) \sum_{a\in\mathcal{A}} \pi(a|s) R(s,a)\right) \\
&= \lim_{T\to\infty} \left(\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \pi(a|s) R(s,a) \sum_{t=1}^T \gamma^{t-1} p(s_t = s|\pi)\right) \\
&= \textstyle\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \pi(a|s) R(s,a) \lim_{T\to\infty} \left(\sum_{t=1}^T \gamma^{t-1} p(s_t = s|\pi)\right) \\
&= \textstyle\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \pi(a|s) R(s,a) d^\pi(s)
\end{aligned} \tag{12}$$

其中，$d^\pi(s) = \sum_{t=1}^\infty \gamma^{t-1} P(s_t = s|\pi)$，也就是在策略$\pi$下状态$s$出现的加权稳态概率。对平均回报也类似，只不过$d^\pi(s) = \lim_{T\to\infty} p(s_t = s|\pi)$，也即采用策略$\pi$的情况下状态$s$出现的稳态概率。

首先我们给一个浅显但是实用的数学技巧：

$$\nabla_\theta \pi(a|s;\theta) = \pi(a|s;\theta) \frac{\nabla_\theta \pi(a|s;\theta)}{\pi(a|s;\theta)} = \pi(a|s;\theta) \nabla_\theta \log \pi(a|s;\theta) \tag{13}$$

我们现在介绍策略梯度算法的基础：策略梯度定理 [8]。

Theorem 4.1. ***(Policy Gradient Theorem)***对任意一个可微的带参策略$\pi(a|s;\theta)$，基于平均回报或折扣回报的策略目标函数$J(\pi;\theta)$有

$$\begin{aligned}\nabla_\theta J(\pi;\theta) &= \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s;\theta) Q^\pi(s,a) \\ &= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} d^\pi(s) \pi(a|s;\theta) \nabla_\theta \log \pi(a|s;\theta) Q^\pi(s,a) \\ &= \mathbb{E}_{(s,a)\sim d^\pi(s)\pi(a|s;\theta)} \left[\nabla_\theta \log \pi(a|s;\theta) Q^\pi(s,a)\right]\end{aligned} \tag{14}$$

由策略梯度定理可到，策略函数的梯度是一个概率期望，从而我们可以采样估计它，然后利用梯度下降算法更新参数，从而收敛到局部最优解。我们注意到策略梯度定理中需要求解$Q^\pi(s,a)$，实践中，我们常常用一个带参函数进行近似

$$Q(s,a;w) \approx Q^\pi(s,a) \Rightarrow \nabla_\theta J(\pi;\theta) \approx \nabla_\theta \log \pi(a|s;\theta) Q(s,a;w)$$

这样我们就有θ和w两个参数需要估计。这也就是所谓的actor-critic算法，critic负责更新w，actor负责更新θ。然而，对$Q(s,a;w)$如果选择不当将导致很高的偏差。

Theorem 4.2. ***(Compatible Function Approximation Theorem)***如果Q_w满足下面两个条件

- 条件一，$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi(a|s;\theta)$。
- 条件二，$w^* = \underset{w}{\operatorname{argmin}} \left[\varepsilon = \mathbb{E}_{(s,a)\sim d^\pi(s)\pi(a|s;\theta)} \left[Q_w(s,a) - Q^\pi(s,a)\right]^2\right]$。

那么就有：$\nabla_\theta J(\pi;\theta) = \mathbb{E}_{(s,a)\sim d^\pi(s)\pi(a|s;\theta)} \left[\nabla_\theta \log \pi(a|s;\theta) Q_w(s,a)\right]$。

这个定理的证明可以参考附录。根据**Theorem4.2**我们可以选择一类函数来近似$Q^\pi(s,a)$使得近似前后梯度不变。

对于确定性策略也有类似的结论 [9]。

$$\begin{aligned}J(\pi;\theta) &= \lim_{T\to\infty} \mathbb{E}\left(\sum_{t=1}^T \gamma^{t-1} r_t|\pi\right) \\ &= \lim_{T\to\infty} \left(\sum_{t=1}^T \gamma^{t-1} \sum_{s\in\mathcal{S}} p(s_t = s|\pi) R(s,\pi(s))\right) \\ &= \lim_{T\to\infty} \left(\sum_{s\in\mathcal{S}} R(s,\pi(s)) \sum_{t=1}^T \gamma^{t-1} p(s_t = s|\pi)\right) \\ &= \sum_{s\in\mathcal{S}} R(s,\pi(s)) d^\pi(s)\end{aligned} \tag{15}$$

Theorem 4.3. ***(Deterministic Policy Gradient Theorem)***对任意一个可微的带参策略$\pi(s;\theta)$，基于平均回报或折扣回报的策略目标函数$J(\pi;\theta)$有

$$\begin{aligned}\nabla_\theta J(\pi;\theta) &= \sum_{s\in\mathcal{S}} d^\pi(s) \nabla_\theta Q^\pi(s,\pi(s;\theta)) \\ &= \mathbb{E}_{s\sim d^\pi(s)} \left[\nabla_\theta \pi(s;\theta) \nabla_a Q^\pi(s,a)\big|_{a=\pi(s;\theta)}\right]\end{aligned} \tag{16}$$

DDPG算法 [10]是近年来颇具代表性的一个策略梯度算法，它有如下特点：

- 是一种在高维状态空间中执行连续控制的算法，也即$\pi(s)$是一个连续值。
- 是一种actor-critic算法。它用一个神经网络$Q(s,a;w)$来近似$Q^\pi(s,a)$，称为critic网络。用另一个函数$\pi(s;\theta)$ 来表示策略，称为actor函数。
- 是确定性策略梯度定理(**Theorem**4.3)的直接运用。
- 采用了类似DQN的手法。首先它也采用双网络策略，有两个critic网络和两个actor网络。其次它也采用了经验回放。

DDPG算法计算过程如**Algorithm**8。

Algorithm 8 DDPG算法.

随机初始化的critic网络$Q(s,a;w)$参数w和actor函数$\pi(s;\theta)$参数θ;
初始化目标critic网络$\widehat{Q}(s,a;\widehat{w})$和目标actor函数$\widehat{\pi}(s;\widehat{\theta})$，$\widehat{w}=w$, $\widehat{\theta}=\theta$;
初始化回放空间D为空;
for 每个episode **do**
    设系统当前状态为s_1。
    for $t\in[1,T]$ **do**
        选择一个动作值$a_t=\pi(s_t;\theta)+\mathcal{N}_t$，$\mathcal{N}_t$是一个随机的噪声;
        执行动作a_t,并得到回报r_t以及系统转移后的新状态s_{t+1};
        把转移样本$d_t=(s_t,a_t,r_t,s_{t+1})$ 存入回放空间D;
        从回放空间D中采样一批转移样本D_t;
        令$error=0$;
        for $d_j=(s_j,a_j,r_j,s_{j+1})\in D_t$ **do**
            $y_j=r_j+\gamma\widehat{Q}(s_{j+1},\widehat{\pi}(s_{j+1};\widehat{\theta});\widehat{w})$
            $error=error+(y_j-Q(s_j,a_j;w))^2$。
        end for
        利用误差$error$更新critic网络Q的参数w。
        利用策略梯度更新actor函数的参数。

$$\nabla_\theta J(\pi;\theta)=\sum_{d_j\in D_t}\nabla_\theta\pi(s;\theta)\big|_{s=s_j}\nabla_a Q^\pi(s,a;w)\big|_{s=s_j,a=\pi(s_j;\theta)}$$

        更新目标critic网络的参数和目标actor函数的参数

$$\widehat{w}=\alpha\widehat{w}+(1-\alpha)w,\quad \widehat{\theta}=\alpha\widehat{\theta}+(1-\alpha)\theta$$

    end for
end for

5 MCTS

MCTS(Monte Carlo Tree Search)是一种在决策空间中进行探索进而发现最优策略的方法，近年来在电脑围棋等机器博弈领域取得了巨大成功。MCTS可以表示成一棵树，每个节点代表一个博弈局面(比如围棋的棋盘)，每个分支代表当前局面下己方可选动作(比如可能的落子位置)。不同于Minimax算法需要计算所有可能局面的博弈值然后才能做出最优决策，MCTS通过采样和模拟来估算每个可能动作的博弈值(这个过程也称为探索Exploration)，然后选择当前博弈值最大的动作予以执行(这个过程也称为利用Exploitation)，接着进行下一轮采样和模拟并更新已有的博弈值使得估算越来越准确。MCTS一次迭代分成4步(图1)：

1. Selection。从根节点出发，根据既定策略不断选择分支往下走，直到达到一个未展开的节点。

2. Expansion。根据可选动作展开当前节点。

3. Simulation。采用默认策略在模拟器中运行，计算出新节点的博弈值。

4. Backpropagation。向上回溯更新父节点的博弈值。

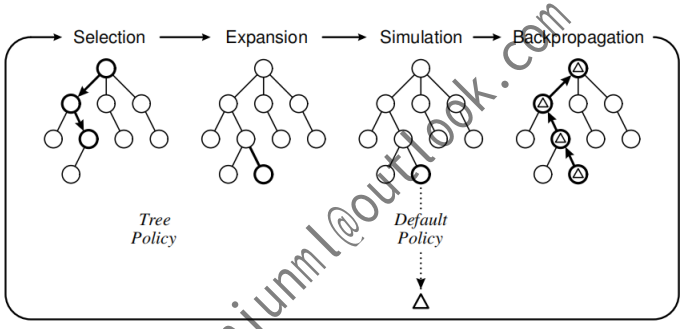

Figure 1: MCTS迭代计算示意图 [11]

UCT [12]是一个很重要的MCTS算法，其计算过程如**Algorithm**9。UCT算法中，每个树节点v有4个属性：该节点对应的博弈局面$s(v)$，从父节点走到本节点所采取的动作$a(v)$，本节点的累计的回报值$Q(v)$，本节点被访问的总次数$N(v)$。$\Delta(v)$表示回报Δ中应该分配节点v的量。

RAVE [13]也是一个比较重要的MCTS算法。

Algorithm 9 UCT算法

function UctSearch(s_0) //UCT算法主函数,参数为状态s_0
    创建相对于状态s_0的根节点v_0。
    while 还有模拟资源 **do**
        $v_l \leftarrow$ TreeSearch(v_0)
        $\Delta \leftarrow$ DefaultPolicy($s(v_l)$)
        Backup(v_l, Δ)
    end while
    return a(BestChild(v_0,0) // 返回根节点汇报值最大的子节点对应的动作
end function

function TreePolicy(v)
    while v不是终结状态 **do**
        if v是未完全展开的节点 **then**
            选择节点v下一个合法且未尝试过的动作a。
            增加节点v的一个子节点v',
            初始化节点v'的信息：$s(v') = f(s(v), a), a(v') = a$。
            return v'
        else
            $v \leftarrow$ BestChild(v,c_p)
        end if
    end while
    return v
end function

function BestChild(v, c)
    return $\underset{v' \in \text{节点}v\text{的子节点}}{\operatorname{argmax}} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2\ln N(v)}{N(v')}}$
end function

function DefaultPolicy(s) // 从状态s开始随机采取动作直到终局
    while s不是终结状态 **do**
        在状态s下随机选择一个动作a。
        $s \leftarrow f(s, a)$
    end while
    return 终结状态s的回报值
end function

function Backup(v, Δ) // 从状态s开始随机采取动作直到终局
    while v非空 **do**
        $N(v) \leftarrow N(v) + 1$。
        $Q(v) \leftarrow Q(v) + \Delta(v)$
        $v \leftarrow v$的父节点
    end while
end function

6 不完全信息博弈

相比于完全信息博弈的博弈树，不完全信息博弈的博弈树有几个不同之处：

- 存在信息集，同一个信息集中的各个博弈状态对博弈者来说无法区分，只能相同的概率分布采取行动，
- 可能存在虚拟的博弈者，其总是以均匀分布采取行动，

博弈者的策略就是计算在其每个信息集上采取每个动作的概率以求达成Nash均衡收益。CFR(Counterfactual Regret Minimization)是不完全信息博弈中著名的一类算法， [14]证明了可以通过分别最小化每个信息集上的Counterfactual regret来最小化全局的regret。

首先介绍一些符号：

- $\mathcal{N} = \{1, 2, \ldots, N\}$ 是博弈者的集合。
- h表示博弈树上的一个节点，
- $A(h)$表示在节点h可以采取的动作集合，
- $\mathcal{I}_i$表示博弈者i的全部信息集，
- $A(I)$表示在信息集I可以采取的动作集合，
- $u_i(z)$表示在叶子节点z处博弈者i的收益，Z表示博弈树上全部的叶子节点，
- σ_i表示博弈者i在其每个信息集上采取行动的概率分布，也称为博弈者i的策略。$\sigma_i(I)(a)$表示博弈者i在信息集I处采取动作a的概率。σ表示全部博弈者的策略，也称游戏的策略概貌。σ_{-i}表示除了博弈者i外其他博弈者的策略。注意，我们只考虑二人博弈，所以$-i$其实就表示i的对手。
- $\pi^\sigma(h)$**表示所有博弈者采取策略**σ**时，整个游戏到达节点**h**的概率。**
- $\pi_i^\sigma(h)$**表示所有博弈者采取策略**σ**时，但是博弈者**i**在每个其信息集总是采取能够到达节点**h**的行动，最后整个游戏到达节点**h **概率。**

$$\pi^\sigma(h) = \prod_i \pi_i^\sigma(h)$$

- $\pi_{-i}^\sigma(h)$**表示反事实达到概率，也就是博弈者**i**以概率1采取能达到节点**h**的动作，同时别的博弈者采用策略**σ_{-i}**，此时整个游戏到达节点**h**的概率。**
- $\pi^\sigma(I)$**表示所有博弈者采取策略**σ**时，信息集**I**的到达概率。**

$$\pi^\sigma(I) = \sum_{h \in I} \pi^\sigma(h)$$

- $\pi_{-i}^\sigma(I)$**表示反事实达到概率，也就是博弈者**i**以概率1采取能达到信息集**I**的动作，同时别的博弈者采用策略**σ_{-i}**，此时整个游戏到达信息集**I**的概率。**
- $\pi^\sigma(h, h')$表示策略概貌为σ时，整个游戏由节点h到节点h'的概率。

- $u_i(\sigma)$表示博弈者i在策略概貌为σ时，其期望收益。

$$u_i(\sigma) = \sum_{z \in Z} u_i(z)\pi^{\sigma}(z)$$

称策略概貌$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ 是一个Nash均衡，当且仅当

$$\text{对}\forall i \in \mathcal{N}, u_i(\sigma) \geq \max_{\sigma_i^{'}} u_i(\sigma_i^{'}, \sigma_{-i})$$

称策略概貌$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ 是一个$\epsilon-$Nash均衡，当且仅当

$$\text{对}\forall i \in \mathcal{N}, u_i(\sigma) + \epsilon \geq \max_{\sigma_i^{'}} u_i(\sigma_i^{'}, \sigma_{-i})$$

现在我们假设博弈进行了多轮，设σ^t表示在第t轮博弈中的策略概貌，T轮博弈后，博弈者i的全局平均regret定义为：

$$R_i^T = \frac{1}{T}\max_{\sigma_i^*} \sum_{t=1}^{T} (u_i(\sigma_i^*, \sigma_{-i}^t) - u_i(\sigma_i^t, \sigma_{-i}^t))$$

T轮博弈后博弈者i的平均策略$\bar{\sigma}_i^T$为：

$$\bar{\sigma}_i^T(I)(a) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma_i^t(I)(a)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)}$$

Theorem 6.1. 在一个零和二人重复博弈中，如果在T轮迭代后，博弈双方的全局平均*regret*满足$R_i^T \leq \epsilon$ ，则$\bar{\sigma}^T$是一个$2\epsilon-$*Nash*均衡。

[15]提供了该定理的一个证明。由定理6.1可知，如果我们最小化每个博弈者的全局平均regret，则我们就能得到一个近似Nash均衡。

首先引进一个新概念，也是CFR算法的核心概念-反事实收益$u_i(\sigma, I)$：

$$u_i(\sigma, I) = \frac{\sum_{h \in I, h' \in Z} \pi_{-i}^{\sigma}(h)\pi^{\sigma}(h, h')u_i(h')}{\pi_{-i}^{\sigma}(I)}$$

定义策略$\sigma|_{I \to a}$为策略概貌为σ，除了在信息集I处对应的博弈者总是采取动作a。接着我们定义即时反事实regret:

$$R_{i,imm}^T(I) = \frac{1}{T}\max_{a \in A(I)} \sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I)(u_i(\sigma^t|_{I \to a}, I) - u_i(\sigma^t, I))$$

通常，我们仅关心正的regret，所以

$$R_{i,imm}^{T,+}(I) = \max(R_{i,imm}^T(I), 0)$$

[14]证明了下面的定理：

Theorem 6.2.

$$R_i^T \leq \sum_{I \in \mathcal{I}_i} R_{i,imm}^{T,+}(I)$$

定理6.2表明可以通过最小化每个信息集上即时反事实regret从而最小化全局平均regret。再结合定理6.1可以得到一个近似的Nash 均衡策略。 [14]还提出可以利用现存的多种Blackwell可近算法(如 [16, 17])在求解每个信息集上的最小化regret策略(也就是每个信息集上的最优动作分布)，如此便得到了一个近似Nash均衡策略。实际上，在 [14]中，$T+1$ 轮中博弈者i在信息集I处采取动作a的概率为:

$$\sigma_i^{T+1}(I)(a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a\in A(I)} R_i^{T,+}(I,a)} & \text{如果} \sum_{a\in A(I)} R_i^{T,+}(I,a) > 0 \\ \frac{1}{|A(I)|} & \text{否则} \end{cases}$$

其中$R_i^{T,+}(I,a) = \max(R_i^T(I,a), 0)$，

$$R_i^T(I,a) = \frac{1}{T}\sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I)(u_i(\sigma^t|_{I\to a}, I) - u_i(\sigma^t, I))$$

7 Adversarial Machine Learning

[18]是第一篇考虑在对抗环境下如何训练分类器的工作，它研究了Naive Bayes分类器中的对抗分类问题，并基于代价敏感学习框架分别为分类器和对抗者找到了一个最优策略的求解算法。对抗者需要对每个样本求一个线性规划问题进而找到修改该样本的哪个属性使得可以被误分且代价最小。[19]则是最近对抗机器学习一篇良好的综述。

[20]提出一种训练可以抵御PGD攻击的神经网络的算法。首先将抵御对抗攻击的神经网络的训练问题形式化如下：

$$\min_\theta \rho(\theta) = E_{(x,y)\sim D}\left[\max_{\delta\in\mathcal{S}} L(x+\delta, y; \theta)\right] \tag{17}$$

其中D表示训练样本集。每个样本x有一个对抗扰动空间$\mathcal{S}$。对于PGD攻击，每个样本x的PGD攻击样本采用如下方式迭代生成：

$$x^{t+1} = \prod_{x+\mathcal{S}} (x^t + \alpha sgn(\nabla_x L(x, y:\theta))) \tag{18}$$

$\prod$表示投影运算。注意著名的用于攻击神经网络的对抗样本生成算法-FGSM [21]是PGD一次迭代形式。求解方程(17)的过程为，对于每个训练样本x，采用方程(18)通过迭代的方式求得一系列x，选择其中使得$L(x+\delta, y;\theta)$局部极大的点x^{ad}，然后采用梯度下降求解下面的问题

$$\underset{\theta}{\text{argmin}} E_{(x,y)\sim D} L(x^{ad}, y; \theta)$$

8 Security Games

- x是leader的混合策略，x_i表示leader采取纯策略i的概率。
- X和Q分别表示leader和follower的纯策略集合。
- q^l表示类型l的follower的策略，这是一个$0-1$向量且是一个单位向量，$q_j^l = 1$则表示follower采取了纯策略j，否则表示没有采取纯策略j。

- R_{ij}^l表示leader采取纯策略i且类型l的follower采取纯策略j时leader的收益。C_{ij}^l则表示此时类型l的follower的收益。
- p^l则表示类型l的follower的出现概率。

9 Team-Maxmin Equilibria

以概率x_s采取纯策略s，采取纯策略s以后，系统的受保护状态为$\Phi(s)$，此时目标i的受保护状态为c_i，那么防御者在该目标上的期望收益为$c_i R_i^d + (1-c_i)P_i^d$。

序列$q_i(x)$表示从博弈树根节点走到节点x的过程中博弈者i所采取的的动作序列。Q_i表示博弈者i的所有可能的动作序列之集合。$r_i(q)$表示序列q被博弈者i执行的概率。

$\chi(I_j^i)$表示在博弈者i的第j个信息集I_j^i中博弈者可以采取的动作集合。Σ_i表示博弈者i的所有可能的动作序列构成的集合。$r_i(\sigma)$表示博弈者i采取动作序列σ的概率。Π_i表示博弈者i在每个信息集上采取某个动作的拼接。$\Delta(\Pi_i)$表示博弈者i在每个信息集上采取每个动作的概率分布的拼接。

这里的博弈模型定义为：同属一个团队$T = \{1, 2, \ldots, n-1\}$的多个博弈者的收益函数在博弈树每个叶子节点都一样$u_i = u_j, \forall i, j \in T$，且他们的目标是最大化各自收益函数之和$u_T = \sum_{i \in T} u_i$，同时使得某个对手$N$的收益函数$u_n$最小化。同时还假设团队内的各个成员独立地采取行动，同时假定是零和博弈$u_n = -u_T$。

Team-Maxmin Equilibrium (TME)就是团队内各个成员独立地确定一个行动策略，同时对手也确定一个行动策略，在此情况下双方都无法通过改变自己的策略获得更大的收益。

$$\underset{r_1, r_2, \ldots, r_{n-1}}{\operatorname{argmax}} \; max_{r_n} \sum_{\sigma} \prod_{i=1}^{n} r_i(\sigma) \sum_{\sigma} U_T(\sigma)$$

10 附录

Theorem 10.1. ***(Compatible Function Approximation Theorem)***如果Q_w满足下面两个条件

- 条件一，$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi(a|s;\theta)$。
- 条件二，$w^* = \underset{w}{\operatorname{argmin}} \left[\varepsilon = \mathbb{E}_{(s,a)\sim d^\pi(s)\pi(a|s;\theta)} \left[Q_w(s,a) - Q^\pi(s,a)\right]^2\right]$。

那么

$$\nabla_\theta J(\pi;\theta) = \mathbb{E}_{(s,a)\sim d^\pi(s)\pi(a|s;\theta)} \left[\nabla_\theta \log \pi(a|s;\theta) Q_w(s,a)\right]$$

我们这里复述 [8]对这个定理的证明(注意，其实这个证明是有问题的)。

Proof. 由条件二可得在最优点w^*有：

$$\nabla_w \varepsilon = \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} d^\pi(s)\pi(a|s;\theta) \left[Q_w(s,a) - Q^\pi(s,a)\right] \nabla_w Q_w(s,a) = 0$$

再结合条件一，可得

$$\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} d^\pi(s)\pi(a|s;\theta) \left[Q_w(s,a) - Q^\pi(s,a)\right] \nabla_\theta \log \pi(a|s;\theta) = 0$$

由策略梯度定理得：

$$\begin{aligned}
\nabla_\theta J(\pi;\theta) &= \textstyle\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} d^\pi(s)\pi(a|s;\theta) \nabla_\theta \log \pi(a|s;\theta) Q^\pi(s,a) - 0 \\
&= \textstyle\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} d^\pi(s)\pi(a|s;\theta) \nabla_\theta \log \pi(a|s;\theta) Q^\pi(s,a) - \\
&\quad \textstyle\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} d^\pi(s)\pi(a|s;\theta) \left[Q_w(s,a) - Q^\pi(s,a)\right] \nabla_\theta \log \pi(a|s;\theta) \\
&= \textstyle\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} d^\pi(s)\pi(a|s;\theta) Q_w(s,a) \nabla_\theta \log \pi(a|s;\theta)
\end{aligned}$$

□

References

[1] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[3] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[4] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[5] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[6] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[7] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.

[8] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[9] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–387–I–395. JMLR.org, 2014.

[10] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[11] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[12] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[13] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artif. Intell.*, 175(11):1856–1875, July 2011.

[14] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2008.

[15] Kevin Waugh and James Andrew Bagnell. A unified view of large-scale zero-sum equilibrium computation. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[16] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 928–936, 2003.

[17] Andrey Bernstein and Nahum Shimkin. Response-based approachability with applications to generalized no-regret problems. *Journal of Machine Learning Research*, 16(24):747–773, 2015.

[18] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM, 2004.

[19] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[20] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[21] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.