附录 B 算法速查表

附录 B 总结了（深度）强化学习的算法和关键概念。这些算法被分为四个部分：深度学习、强化学习、深度强化学习和高等深度强化学习。为了便于读者学习，我们为每个算法提供了伪代码。我们尽量在行文中保持数学符号、变量记号和术语与整本书一致。

B.1 深度学习

B.1.1 随机梯度下降

算法 B.43 随机梯度下降的训练过程

Input: 参数 $\boldsymbol{\theta}$, 学习率 α, 训练步数/迭代次数 S

for $i = 0$ **to** S **do**

计算一个小批量的 $\mathcal{L}$

通过反向传播计算 $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$

$\nabla \boldsymbol{\theta} \leftarrow -\alpha \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$;

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \nabla \boldsymbol{\theta}$ 更新参数

end for

return $\boldsymbol{\theta}$; 返回训练好的参数

B.1.2　Adam 优化器

算法 B.44 Adam 优化器的训练过程

Input: 参数 $\boldsymbol{\theta}$, 学习率 α, 训练步数/迭代次数 S, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

$\boldsymbol{m}_0 \leftarrow 0$; 初始化一阶动量

$\boldsymbol{v}_0 \leftarrow 0$; 初始化二阶动量

for $t = 1$ **to** S **do**

$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$; 用一个随机的小批次计算梯度

$\boldsymbol{m}_t \leftarrow \beta_1 * \boldsymbol{m}_{t-1} + (1 - \beta_1) * \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$; 更新一阶动量

$\boldsymbol{v}_t \leftarrow \beta_2 * \boldsymbol{v}_{t-1} + (1 - \beta_2) * (\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}})^2$; 更新二阶动量

$\hat{\boldsymbol{m}}_t \leftarrow \frac{\boldsymbol{m}_t}{1-\beta_1^t}$; 计算一阶动量的滑动平均

$\hat{\boldsymbol{v}}_t \leftarrow \frac{\boldsymbol{v}_t}{1-\beta_2^t}$; 计算二阶动量的滑动平均

$\nabla \boldsymbol{\theta} \leftarrow -\alpha * \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \nabla \boldsymbol{\theta}$; 更新参数

end for

return $\boldsymbol{\theta}$; 返回训练好的参数

B.2　强化学习

B.2.1　赌博机

随机多臂赌博机（Stochastic Multi-armed Bandit）

算法 B.45 多臂赌博机学习

初始化 K 个手臂

定义总时长为 T

每一个手臂都有一个对应的 $v_i \in [0, 1]$. 每一个奖励都是独立同分布地从 v_i 中采样得到的

for $t = 1, 2, \cdots, T$ **do**

智能体从 K 个手臂中选择 $A_t = i$

环境返回奖励值向量 $R_t = (R_t^1, R_t^2, \cdots, R_t^K)$

智能体观测到 R_t^i

end for

对抗多臂赌博机（Adversarial Multi-armed Bandit）

算法 B.46 对抗多臂赌博机

初始化 K 个机器手臂

for $t = 1, 2, \cdots, T$ **do**
智能体在 K 个手臂当中选中 I_t
对抗者选择一个奖励值向量 $\boldsymbol{R}_t = (R_t^1, R_t^2, \cdots, R_t^K) \in [0,1]^K$
智能体观察到奖励 $R_t^{I_t}$（根据具体的情况也有可能看到整个奖励值向量）
end for

算法 B.47 针对对抗多臂赌博机的 Hedge 算法

初始化 K 个手臂
$G_i(0)$ for $i = 1, 2, \cdots, K$
for $t = 1, 2, \cdots, T$ **do**
智能体从 $p(t)$ 分布中选择 $A_t = i_t$，其中

$$p_i(t) = \frac{\exp(\eta G_i(t-1))}{\sum_j^K \exp(\eta G_j(t-1))}$$

智能体观测到奖励 g_t
让 $G_i(t) = G(t-1) + g_t^i,\ \forall i \in [1, K]$
end for

B.2.2 动态规划

策略迭代（Policy Iteration）

算法 B.48 策略迭代

对于所有的状态初始化 V 和 π
repeat
//执行策略评估
repeat
$\delta \leftarrow 0$
for $s \in \mathcal{S}$ **do**
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_{r,s'}(r + \gamma V(s'))P(r, s'|s, \pi(s))$
$\delta \leftarrow \max(\delta, |v - V(s)|)$
end for
until δ 小于一个正阈值
//执行策略提升
stable $\leftarrow$ true

for $s \in \mathcal{S}$ **do**
$a \leftarrow \pi(s)$
$\pi(s) \leftarrow \arg\max_a \sum_{r,s'}(r + \gamma V(s'))P(r, s'|s, a)$
if $a \neq \pi(s)$ **then**
stable $\leftarrow$ false
end if
end for
until stable $=$ true
return 策略 π

价值迭代（Value Iteration）

算法 B.49 价值迭代

为所有状态初始化 V
repeat
$\delta \leftarrow 0$
for $s \in \mathcal{S}$ **do**
$u \leftarrow V(s)$
$V(s) \leftarrow \max_a \sum_{r,s'} P(r, s'|s, a)(r + \gamma V(s'))$
$\delta \leftarrow \max(\delta, |u - V(s)|)$
end for
until δ 小于一个正阈值
输出贪心策略 $\pi(s) = \arg\max_a \sum_{r,s'} P(r, s'|s, a)(r + \gamma V(s'))$

B.2.3　蒙特卡罗

蒙特卡罗预测

算法 B.50 首次蒙特卡罗预测

输入：初始化策略 π
初始化所有状态的 $V(s)$
初始化一列回报：Returns(s) 对所有状态
repeat
通过 π: $S_0, A_0, R_0, S_1, \cdots, S_{T-1}, A_{T-1}, R_t$ 生成一个回合
$G \leftarrow 0$
$t \leftarrow T - 1$
for $t >= 0$ **do**
$G \leftarrow \gamma G + R_{t+1}$

if $S_0, S_1, \cdots, S_{t-1}$ 没有 S_t **then**
Returns(S_t).append(G)
$V(S_t) \leftarrow$ mean(Returns(S_t))
end if
$t \leftarrow t-1$
end for
until 收敛

蒙特卡罗控制

算法 B.51 蒙特卡罗探索开始

初始化所有状态的 $\pi(s)$
对于所有的状态-动作对，初始化 $Q(s,a)$ 和 Returns(s,a)
repeat
随机选择 S_0 和 A_0 直到所有状态-动作对的概率为非零
根据 π: $S_0, A_0, R_0, S_1, \cdots, S_{T-1}, A_{T-1}, R_t$ 来生成 S_0, A_0
$G \leftarrow 0$
$t \leftarrow T-1$
for $t >= 0$ **do**
$G \leftarrow \gamma G + R_{t+1}$
if $S_0, A_0, S_1, A_1 \cdots, S_{t-1}, A_{t-1}$ 没有 S_t, A_t **then**
Returns(S_t, A_t).append(G)
$Q(S_t, A_t) \leftarrow$ mean(Returns(S_t, A_t))
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
end if
$t \leftarrow t-1$
end for
until 收敛

时间差分（Temporal Difference，TD）

算法 B.52 TD(0) 对状态值的估算

输入策略 π
初始化 $V(s)$ 和步长 $\alpha \in (0,1]$
for 每一个回合 **do**
初始化 S_0
for 每一个在现有的回合的 S_t **do**
$A_t \leftarrow \pi(S_t)$

$R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
end for
end for

TD(λ)

算法 B.53 状态值半梯度 TD(λ)

输入策略 π
初始化一个可求导的状态值函数 v、步长 α 和状态值函数权重 $\boldsymbol{w}$
for 对每一个回合 **do**
初始化 S_0
$\boldsymbol{z} \leftarrow 0$
for 每一个本回合的步骤 S_t **do**
使用 π 来选择 A_t
$R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
$\boldsymbol{z} \leftarrow \gamma\lambda\boldsymbol{z} + \nabla V(S_t, \boldsymbol{w}_t)$
$\delta \leftarrow R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{w}_t) - V(S_t, \boldsymbol{w}_t)$
$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\delta\boldsymbol{z}$
end for
end for

Sarsa：在线策略 TD 控制

算法 B.54 Sarsa （在线策略 TD 控制）

对所有的状态-动作对初始化 $Q(s, a)$
for 每一个回合 **do**
初始化 S_0
用一个基于 Q 的策略来选择 A_0
for 每一个在当前回合的 S_t **do**
用一个基于 Q 的策略从 S_t 选择 A_t
$R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
从 S_{t+1} 中用一个基于 Q 的策略来选择 A_{t+1}
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
end for
end for

N 步 Sarsa

算法 B.55 N 步 Sarsa

对所有的状态动作对初始化 $Q(s,a)$
初始化步长 $\alpha \in (0,1]$
决定一个固定的策略 π 或者使用 ϵ-贪心
for 每一个回合 **do**
  初始化 S_0
  使用 $\pi(S_0,A)$ 来选择 A_0
  $T \leftarrow$ INTMAX （一个回合的长度）
  $\gamma \leftarrow 0$
  for $t \leftarrow 0,1,2,\cdots$ until $\gamma - T - 1$ **do**
    if $t < T$ **then**
      $R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
      if S_{t+1} 是终止状态 **then**
        $T \leftarrow t+1$
      else
        使用 $\pi(S_t,A)$ 来选择 A_{t+1}
      end if
    end if
    $\tau \leftarrow t-n+1$ （更新的时间点。这个是 n 步 Sarsa，所以只需要更新那个 $n+1$ 前的一步，就会持续这样下去，直到所有状态都被更新。）
    if $\tau \geqslant 0$ **then**
      $G \leftarrow \sum_{i=\tau+1}^{\min(r+n,T)} \gamma^{i-\gamma-1} R_i$
      if $\gamma + n < T$ **then**
        $G \leftarrow G + \gamma^n Q(S_{t+n}, A_{\gamma+n})$
      end if
      $Q(S_\gamma, A_\gamma) \leftarrow Q(S_\gamma, A_\gamma) + \alpha[G - Q(S_\gamma, A_\gamma)]$
    end if
  end for
end for

Q-learning：离线策略 TD 控制

算法 B.56 Q-learning （离线策略 TD 控制）

初始化所有的状态-动作对的 $Q(s,a)$ 以及步长 $\alpha \in (0,1]$
for 每一个回合 **do**

初始化 S_0
for 每一个在当前回合的 S_t **do**
使用基于 Q 的策略来选择 A_t
$R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
end for
end for

B.3　深度强化学习

深度 Q 网络（Deep Q-Networks，DQN）是一个将 Q-learning 通过深度神经网络来拟合价值函数，从而延伸到高维情况的方法，它使用一个目标动作价值网络和一个经验回放缓存来更新。

主要思想：

- 用神经网络进行 Q 值函数拟合；
- 用经验回放缓存进行离线更新；
- 目标网络和延迟更新；
- 用均方误差或 Huber 损失来最小化时间差分（Temporal Difference，TD）误差。

算法 B.57 DQN

超参数: 回放缓存容量 N、奖励折扣因子 γ、用于目标状态-动作值函数更新的延迟步长 C、ϵ-greedy 中的 ϵ
输入: 空回放缓存 $\mathcal{D}$，初始化状态-动作值函数 Q 的参数 θ
使用参数 $\hat{\theta} \leftarrow \theta$ 初始化目标状态-动作值函数 $\hat{Q}$
for 片段 $= 0, 1, 2, \cdots$ **do**
初始化环境并获取观测数据 O_0
初始化序列 $S_0 = \{O_0\}$ 并对序列进行预处理 $\phi_0 = \phi(S_0)$
for t $= 0, 1, 2, \cdots$ **do**
通过概率 ϵ 选择一个随机动作 A_t，否则选择动作 $A_t = \arg\max_a Q(\phi(S_t), a; \theta)$
执行动作 A_t 并获得观测数据 O_{t+1} 和奖励数据 R_t
如果本局结束，则设置 $D_t = 1$，否则 $D_t = 0$
设置 $S_{t+1} = \{S_t, A_t, O_{t+1}\}$ 并进行预处理 $\phi_{t+1} = \phi(S_{t+1})$
存储状态转移数据 $(\phi_t, A_t, R_t, D_t, \phi_{t+1})$ 到 $\mathcal{D}$ 中
从 $\mathcal{D}$ 中随机采样小批量状态转移数据 $(\phi_i, A_i, R_i, D_i, \phi'_i)$
如果 $D_i = 0$，设置 $Y_i = R_i + \gamma \max_{a'} \hat{Q}(\phi'_i, a'; \hat{\theta})$，否则设置 $Y_i = R_i$
在 $(Y_i - Q(\phi_i, A_i; \theta))^2$ 上对 θ 执行梯度下降步骤

每 C 步对目标网络 $\hat{Q}$ 进行同步

如果片段结束，则跳出循环

end for

end for

Double DQN 是一个 DQN 的改进版本，用来解决过估计（Overestimation）问题。

主要思想：

- 双 Q 网络是一种对目标价值估计的嵌入式方法，一个 Q 估计值被嵌入另一个 Q 估计值中。

更改上面 DQN 算法的第 14 行为令 $Y_j = R_j + \gamma(1-D_j)\hat{Q}(\phi_{j+1}, \arg\max_{a'} Q(\phi_{j+1}, a'; \theta_j); \hat{\theta})$。

Dueling DQN 是对 DQN 的一个改进版本，它将动作价值函数分解为一个状态价值函数和一个依赖状态的动作优势函数。

主要思想：

- 将动作价值函数 Q 分解为值函数 V 和优势函数 A。

更改 DQN 中动作价值函数 Q（及它的目标 $\hat{Q}$）的参数化方式为 $Q(s,a;\theta,\theta_v,\theta_a) = V(s;\theta,\theta_v) + (A(s,a;\theta,\theta_a) - \max_{a'} A(s,a';\theta,\theta_a))$ 或，$Q(s,a;\theta,\theta_v,\theta_a) = V(s;\theta,\theta_v) + (A(s,a;\theta,\theta_a) - \frac{1}{|\mathcal{A}|} A(s,a';\theta,\theta_a))$。

REINFORCE 是一个使用基于策略优化和在线策略更新的算法。

算法 B.58 REINFORCE

输入: 初始策略参数 θ

for $\mathrm{k} = 0, 1, 2, \cdots$ **do**

初始化环境

通过在环境中运行策略 $\pi_k = \pi(\theta_k)$ 收集轨迹数据集 $\mathcal{D}_k = \{\tau_i = \{(S_t, A_t, R_t) | t = 0, 1, \cdots, T\}\}$

计算累计奖励 G_t

估计策略梯度 $g_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t)|_{\theta_k} G_t$

通过梯度上升更新策略 $\theta_{k+1} = \theta_k + \alpha_k g_k$

end for

带基准函数的 REINFORCE 算法或称**初版策略梯度**（REINFORCE with Baseline/Vanilla Policy Gradient）是 REINFORCE 的另一个版本，它使用动作优势函数而不是累计奖励来估计策略梯度。

算法 B.59 带基准函数的 REINFORCE 算法

超参数: 步长 η_θ、奖励折扣因子 γ、总步数 L、批尺寸 B、基准函数 b。

输入: 初始策略参数 θ_0

初始化 $\theta = \theta_0$

for $k = 1, 2, \cdots,$ **do**

执行策略 π_θ 得到 B 个轨迹，每一个有 L 步，并收集 $\{S_{t,\ell}, A_{t,\ell}, R_{t,\ell}\}$。

$\hat{A}_{t,\ell} = \sum_{\ell'=\ell}^{L} \gamma^{\ell'-\ell} R_{t,\ell} - b(S_{t,\ell})$

$J(\theta) = \frac{1}{B}\sum_{t=1}^{B}\sum_{\ell=0}^{L}\log\pi_\theta(A_{t,\ell}|S_{t,\ell})\hat{A}_{t,\ell}$
$\theta = \theta + \eta_\theta\nabla J(\theta)$
用 $\{S_{t,\ell}, A_{t,\ell}, R_{t,\ell}\}$ 更新 $b(S_{t,\ell})$
end for
返回 θ

Actor-Critic 是一个改自 REINFORCE 的算法，它使用价值函数拟合。

算法 B.60 Actor-Critic 算法

超参数: 步长 η_θ 和 η_ψ、奖励折扣因子 γ
输入: 初始策略函数参数 θ_0, 初始价值函数参数 ψ_0
初始化 $\theta = \theta_0$ 和 $\psi = \psi_0$
for $t = 0, 1, 2, \cdots$ **do**
执行一步策略 π_θ, 保存 $\{S_t, A_t, R_t, S_{t+1}\}$
估计优势函数 $\hat{A}_t = R_t + \gamma V_\psi^{\pi_\theta}(S_{t+1}) - V_\psi^{\pi_\theta}(S_t)$
$J(\theta) = \sum_t \log\pi_\theta(A_t|S_t)\hat{A}_t$
$J_{V_\psi^{\pi_\theta}}(\psi) = \sum_t \hat{A}_t^2$
$\psi = \psi + \eta_\psi\nabla J_{V_\psi^{\pi_\theta}}(\psi)$, $\theta = \theta + \eta_\theta\nabla J(\theta)$
end for
返回 (θ, ψ)

Q 值 Actor-Critic（Q-value Actor-Critic，QAC）是另一个版本的 Actor-Critic 算法，作为基于价值（比如 Q-Learning）和基于策略（比如 REINFORCE）优化方法的结合，使用在线策略更新的方式。

主要思想：

- 结合 DQN 和 REINFORCE。

算法 B.61 QAC

输入: 初始策略参数 θ、初始动作价值函数 Q 的参数 ω、折扣因子 γ
for k $= 0, 1, 2, \cdots$ **do**
初始化环境
通过在环境中运行策略 $\pi_k = \pi(\theta_k)$，收集轨迹数据集 $\mathcal{D}_k = \{\tau_i = \{(S_t, A_t, R_t, D_t)|t = 0, 1, \cdots, T\}\}$。
计算 TD 误差 $\delta_t = R_t + \gamma\max_{a'} Q_\omega(S_{t+1}, a') - Q_\omega(S_t, A_t)$
估计策略梯度如 $g_k = \frac{1}{|\mathcal{D}_k|}\sum_{\tau\in\mathcal{D}_k}\sum_{t=0}^{\mathrm{T}}\nabla_\theta\log\pi_\theta(A_t|S_t)|_{\theta_k}Q_\omega(S_t, A_t)$
通过梯度上升更新策略 $\theta_{k+1} = \theta_k + \alpha_k g_k$
使用均方误差更新动作价值函数 $\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T}\sum_{\tau\in\mathcal{D}_k}\sum_{t=0}^{\mathrm{T}}\delta_t^2$ 通过梯度下降算法

end for

优势 Actor-Critic（Advantage Actor-Critic，A2C）是 Actor-Critic 算法的改进版本，它使用有基准的 REINFORCE 而非初版 REINFORCE 来进行策略优化，并且使用在线策略更新。

主要思想：

- 结合 DQN 和有基准的 REINFORCE。

算法 B.62 A2C

Master:

超参数: 步长 η_ψ 和 η_θ, worker 节点集 $\mathcal{W}$

输入: 初始策略函数参数 θ_0, 初始价值函数参数 ψ_0

初始化 $\theta = \theta_0$ 和 $\psi = \psi_0$

for $k = 0, 1, 2, \cdots$ **do**

$(g_\psi, g_\theta) = 0$

for $\mathcal{W}$ 里每一个 worker 节点 **do**

$(g_\psi, g_\theta) = (g_\psi, g_\theta) + \textbf{worker}(V_\psi^{\pi_\theta}, \pi_\theta)$

end for

$\psi = \psi - \eta_\psi g_\psi$; $\theta = \theta + \eta_\theta g_\theta$。

end for

Worker:

超参数: 奖励折扣因子 γ, 轨迹长度 L

输入: 价值函数 $V_\psi^{\pi_\theta}$, 策略函数 π_θ

执行 L 步策略 π_θ, 保存 $\{S_t, A_t, R_t, S_{t+1}\}$

估计优势函数 $\hat{A}_t = R_t + \gamma V_\psi^{\pi_\theta}(S_{t+1}) - V_\psi^{\pi_\theta}(S_t)$

$J(\theta) = \sum_t \log \pi_\theta(A_t|S_t)\hat{A}_t$

$J_{V_\psi^{\pi_\theta}}(\psi) = \sum_t \hat{A}_t^2$

$(g_\psi, g_\theta) = (\nabla J_{V_\psi^{\pi_\theta}}(\psi), \nabla J(\theta))$

返回 (g_ψ, g_θ)

异步优势 Actor-Critic（Asynchronous Advantage Actor-Critic，A3C）是一个 A2C 的修改版本，它使用异步梯度更新来实现大规模并行计算。

主要思想：

- 异步更新策略。

算法 B.63 A3C

Master:

超参数: 步长 η_ψ 和 η_θ, 当前策略函数 π_θ, 价值函数 $V_\psi^{\pi_\theta}$

输入: 梯度 g_ψ, g_θ

$\psi = \psi - \eta_\psi g_\psi; \theta = \theta + \eta_\theta g_\theta$。

返回 $(V_\psi^{\pi_\theta}, \pi_\theta)$

Worker:

超参数: 奖励折扣因子 γ、轨迹长度 L

输入: 策略函数 π_θ、价值函数 $V_\psi^{\pi_\theta}$

$(g_\theta, g_\psi) = (0, 0)$

for $k = 1, 2, \cdots,$ **do**

$(\theta, \psi) =$ **Master**(g_θ, g_ψ)

执行 L 步策略 π_θ, 保存 $\{S_t, A_t, R_t, S_{t+1}\}$。

估计优势函数 $\hat{A}_t = R_t + \gamma V_\psi^{\pi_\theta}(S_{t+1}) - V_\psi^{\pi_\theta}(S_t)$

$J(\theta) = \sum_t \log \pi_\theta(A_t|S_t)\hat{A}_t$

$J_{V_\psi^{\pi_\theta}}(\psi) = \sum_t \hat{A}_t^2$

$(g_\psi, g_\theta) = (\nabla J_{V_\psi^{\pi_\theta}}(\psi), \nabla J(\theta))$

end for

深度确定性策略梯度（Deep Deterministic Policy Gradient，DDPG）是 DQN 和 QAC 的结合，它使用确定性策略，并采用经验回放缓存和离线策略更新的方式。

主要思想：

- 确定性策略作为动作空间上 Q 值的最大化算子的拟合；
- 用 Ornstein-Uhlenbeck 或高斯噪声进行随机动作的探索；
- 目标网络和延迟更新。

算法 B.64 DDPG

超参数：软更新因子 ρ，奖励折扣因子 γ

输入：回放缓存 $\mathcal{D}$，初始化 Critic 网络 $Q(s, a|\theta^Q)$ 参数 θ^Q、Actor 网络 $\pi(s|\theta^\pi)$ 参数 θ^π、目标网络 Q'、π'

初始化目标网络参数 Q' 和 π'，赋值 $\theta^{Q'} \leftarrow \theta^Q, \theta^{\pi'} \leftarrow \theta^\pi$

for episode $= 1, M$ **do**

初始化随机过程 $\mathcal{N}$ 用于给动作添加探索

接收初始状态 S_1

for t $= 1, T$ **do**

选择动作 $A_t = \pi(S_t|\theta^\pi) + \mathcal{N}_t$

执行动作 A_t 得到奖励 R_t，转移到下一状态 S_{t+1}

存储状态转移数据对 $(S_t, A_t, R_t, D_t, S_{t+1})$ 到 $\mathcal{D}$

令 $Y_i = R_i + \gamma(1-D_t)Q'(S_{t+1}, \pi'(S_{t+1}|\theta^{\pi'})|\theta^{Q'})$

通过最小化损失函数更新 Critic 网络：

$L = \frac{1}{N}\sum_i (Y_i - Q(S_i, A_i|\theta^Q))^2$

通过策略梯度的方式更新 Actor 网络：

$\nabla_{\theta^\pi} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=S_i, a=\pi(S_i)} \nabla_{\theta^\pi}\pi(s|\theta^\pi)|_{S_i}$

更新目标网络：

$\theta^{Q'} \leftarrow \rho\theta^Q + (1-\rho)\theta^{Q'}$

$\theta^{\pi'} \leftarrow \rho\theta^\pi + (1-\rho)\theta^{\pi'}$

end for

end for

孪生延迟 DDPG（Twin Delayed DDPG，TD3）是一个更先进的基于 DDPG 的算法，它使用孪生动作价值网络，并对策略和目标网络采用延迟更新。

主要思想：

- Double Q-learning；
- 对目标网络和策略的延迟更新；
- 对目标策略的平滑正则化。

算法 B.65 TD3

超参数：软更新因子 ρ，回报折扣因子 γ，截断因子 c

输入：回放缓存 $\mathcal{D}$，初始化 Critic 网络 $Q_{\theta_1}, Q_{\theta_2}$ 参数 θ_1, θ_2，初始化 Actor 网络 π_ϕ 参数 ϕ

初始化目标网络参数 $\hat{\theta}_1 \leftarrow \theta_1, \hat{\theta}_2 \leftarrow \theta_2, \hat{\phi} \leftarrow \phi$

for $t = 1$ to T do **do**

选择动作 $A_t \sim \pi_\phi(S_t) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$

接受奖励 R_t 和新状态 S_{t+1}

存储状态转移数据对 $(S_t, A_t, R_t, D_t, S_{t+1})$ 到 $\mathcal{D}$

从 $\mathcal{D}$ 中采样大小为 N 的小批量样本 $(S_t, A_t, R_t, D_t, S_{t+1})$

$\tilde{a}_{t+1} \leftarrow \pi_{\phi'}(S_{t+1}) + \epsilon, \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}, -c, c))$。

$y \leftarrow R_t + \gamma(1-D_t)\min_{i=1,2} Q_{\theta_i'}(S_{t+1}, \tilde{a}_{t+1})$

更新 Critic 网络 $\theta_i \leftarrow \arg\min_{\theta_i} N^{-1}\sum (y - Q_{\theta_i}(S_t, A_t))^2$

if t mod d **then**

更新 ϕ：

$\nabla_\phi J(\phi) = N^{-1}\sum \nabla_a Q_{\theta_1}(S_t, A_t)|_{A_t=\pi_\phi(S_t)} \nabla_\phi \pi_\phi(S_t)$

更新目标网络：

$\hat{\theta}_i \leftarrow \rho\theta_i + (1-\rho)\hat{\theta}_i$

$\hat{\phi} \leftarrow \rho\phi + (1-\rho)\hat{\phi}$

end if

end for

柔性 Actor-Critic（Soft Actor-Critic，SAC）是一个更先进的基于 DDPG 的算法，使用额外的柔性熵（Soft Entropy）项来促进探索。

主要思想：

- 熵正则化来促进探索；
- Double Q-learning；
- 再参数化技巧使得随机性策略可微并用确定性策略梯度更新；
- Tanh 高斯型动作分布。

算法 B.66 SAC

超参数: 目标熵 κ, 步长 $\lambda_Q, \lambda_\pi, \lambda_\alpha$, 指数移动平均系数 τ

输入: 初始策略函数参数 θ, 初始 Q 值函数参数 ϕ_1 及 ϕ_2

$\mathcal{D} = \emptyset$; $\tilde{\phi}_i = \phi_i$, for $i = 1, 2$

for $k = 0, 1, 2, \cdots$。 **do**

  for $t = 0, 1, 2, \cdots$ **do**

    从 $\pi_\theta(\cdot|S_t)$ 中取样 A_t, 保存 (R_t, S_{t+1})。

    $\mathcal{D} = \mathcal{D} \cup \{S_t, A_t, R_t, S_{t+1}\}$

  end for

  进行多步梯度更新：

$$\phi_i = \phi_i - \lambda_Q \nabla J_Q(\phi_i) \text{ for } i = 1, 2$$

$$\theta = \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$$

$$\alpha = \alpha - \lambda_\alpha \nabla J(\alpha)$$

$$\tilde{\phi}_i = (1 - \tau)\phi_i + \tau\tilde{\phi}_i \text{ for } i = 1, 2$$

end for

返回 θ, ϕ_1, ϕ_2。

信赖域策略优化（Trust Region Policy Optimization，TRPO）是一个使用二阶梯度下降和在线策略更新的信赖域算法。

主要思想：

- 用 KL 散度（KL-divergence）来使得新旧策略在策略空间中接近；
- 有限制的二阶优化方法；
- 使用共轭梯度（Conjugate Gradient）来避免计算逆矩阵（Inverse Matrix）。

算法 B.67 TRPO

超参数: KL-散度上限 δ, 回溯系数 α, 最大回溯步数 K

输入: 回放缓存 $\mathcal{D}_k$, 初始策略函数参数 θ_0, 初始价值函数参数 ϕ_0

for episode $= 0, 1, 2, \cdots$ **do**

在环境中执行策略 $\pi_k = \pi(\boldsymbol{\theta}_k)$ 并保存轨迹集 $\mathcal{D}_k = \{\tau_i\}$

计算将得到的奖励 $\hat{G}_t$

基于当前的价值函数 V_{ϕ_k} 计算优势函数估计 $\hat{A}_t$ （使用任何估计优势的方法）

估计策略梯度 $\hat{\boldsymbol{g}}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t)\big|_{\boldsymbol{\theta}_k} \hat{A}'_t$

使用共轭梯度算法计算 $\hat{\boldsymbol{x}}_k \approx \hat{\boldsymbol{H}}_k^{-1} \hat{\boldsymbol{g}}_k$ 这里 $\hat{\boldsymbol{H}}_k$ 是样本平均 KL 散度的 Hessian 矩阵

通过回溯线搜索更新策略 $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha^j \sqrt{\frac{2\delta}{\hat{\boldsymbol{x}}_k^{\mathrm{T}} \hat{\boldsymbol{H}}_k \hat{\boldsymbol{x}}_k}} \hat{\boldsymbol{x}}_k$ 这里 j 是 $\{0, 1, 2, \cdots K\}$ 中提高样本损失并且满足样本 KL 散度约束的最小值

通过使用梯度下降的算法最小化均方误差来拟合价值函数：$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{\mathrm{T}} \left(V_\phi(S_t) - \hat{G}_t\right)^2$

end for

近端策略优化（惩罚型）（Proximal Policy Optimization，PPO-Penalty）是一个基于 TRPO 的信赖域算法，它使用一阶梯度和以一个自适应惩罚项实现的信赖域限制。

主要思想：

- 用 KL 散度来使得新旧策略在策略空间中接近；
- 将受限优化问题转化为一个不受限的问题；
- 用一阶方法来避免计算 Hessian 矩阵；
- 自适应地调整惩罚系数。

算法 B.68 PPO-Penalty

超参数: 奖励折扣因子 γ，KL 散度惩罚系数 λ，适应性参数 $a = 1.5, b = 2$, 子迭代次数 M, B。

输入: 初始策略函数参数 θ、初始价值函数参数 ϕ。

for k $= 0, 1, 2, \cdots$ **do**

执行 T 步策略 π_θ，保存 $\{S_t, A_t, R_t\}$。

估计优势函数 $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} R_{t'} - V_\phi(S_t)$。

$\pi_{\text{old}} \leftarrow \pi_\theta$

for $m \in \{1, \cdots, M\}$ **do**

$J_{\text{PPO}}(\theta) = \sum_{t=1}^{\mathrm{T}} \frac{\pi_\theta(A_t|S_t)}{\pi_{\text{old}}(A_t|S_t)} \hat{A}_t - \lambda \hat{\mathbb{E}}_t \left[D_{\text{KL}}(\pi_{\text{old}}(\cdot|S_t) \| \pi_\theta(\cdot|S_t))\right]$

使用梯度算法基于 $J_{\text{PPO}}(\theta)$ 更新策略函数参数 θ。

end for

for $b \in \{1, \cdots, B\}$ **do**

$L(\phi) = -\sum_{t=1}^{\mathrm{T}} \left(\sum_{t'>t} \gamma^{t'-t} R_{t'} - V_\phi(S_t)\right)^2$

使用梯度算法基于 $L(\phi)$ 更新价值函数参数 ϕ。

end for

计算 $d=\hat{\mathbb{E}}_t\left[D_{\mathrm{KL}}(\pi_{\mathrm{old}}(\cdot|S_t)\|\pi_\theta(\cdot|S_t))\right]$

if $d<d_{\text{target}}/a$ **then**

$\lambda\leftarrow\lambda/b$

else if $d>d_{\text{target}}\times a$ **then**

$\lambda\leftarrow\lambda\times b$

end if

end for

近端策略优化（截断型）是一个基于 TRPO 的信赖域算法，它使用一阶梯度和以一个对梯度的截断方法实现的信赖域限制。

主要思想：

- 在目标函数中用截断方法替换 KL-散度的限制。

算法 B.69 PPO-Clip

超参数: 截断因子 ϵ, 子迭代次数 M,B。

输入: 初始策略函数参数 θ, 初始价值函数参数 ϕ

for $\mathrm{k}=0,1,2,\cdots$ **do**

在环境中执行策略 π_{θ_k} 并保存轨迹集 $\mathcal{D}_k=\{\tau_i\}$

计算将得到的奖励 $\hat{G}_t$

基于当前的价值函数 V_{ϕ_k} 计算优势函数 $\hat{A}_t$（基于任何优势函数的估计方法）

for $m\in\{1,\cdots,M\}$ **do**

$\ell_t(\theta')=\frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_{\text{old}}}(A_t|S_t)}$ 采用 Adam 随机梯度上升算法最大化 PPO-Clip 的目标函数来更新策略：

$$\theta_{k+1}=\arg\max_\theta\frac{1}{|\mathcal{D}_k|T}\sum_{\tau\in D_k}\sum_{t=0}^{\mathrm{T}}\min(\ell_t(\theta')A^{\pi_{\theta_{\text{old}}}}(S_t,A_t),$$

$$\mathrm{clip}(\ell_t(\theta'),1-\epsilon,1+\epsilon)A^{\pi_{\theta_{\text{old}}}}(S_t,A_t))$$

end for

for $b\in\{1,\cdots,B\}$ **do**

采用梯度下降方法最小化均方误差来学习价值函数:

$\phi_{k+1}=\arg\min_\phi\frac{1}{|\mathcal{D}_k|T}\sum_{\tau\in\mathcal{D}_k}\sum_{t=0}^{\mathrm{T}}\left(V_\phi(S_t)-\hat{G}_t\right)^2$

end for

end for

使用 Kronecker 因子化信赖域的 Actor-Critic（Actor Critic using Kronecker-Factored Trust Region，ACKTR）是一种信赖域在线策略算法，对二阶自然梯度计算使用 Kronecker 因子化近似。

主要思想：

- 使用自然梯度的二阶优化；
- 对自然梯度进行 K-FAC 近似。

算法 B.70 ACKTR

超参数: 步长 $\eta_{\max}$, KL-散度上限 δ

输入: 空回放缓存 $\mathcal{D}$, 初始策略函数参数 θ_0, 初始价值函数参数 ϕ_0

for k $= 0, 1, 2, \cdots$ **do**

在环境中执行策略 $\pi_k = \pi(\theta_k)$ 并保存轨迹集 $\mathcal{D}_k = \{\tau_i | i = 0, 1, \cdots\}$

计算累积奖励 G_t

基于当前的价值函数 V_{ϕ_k} 计算优势函数 $\hat{A}_t$（基于任何优势函数的估计方法）

估计策略梯度 $\hat{\boldsymbol{g}}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t)\big|_{\theta_k} \hat{A}_t$

for $l = 0, 1, 2, \cdots$ **do**

$\mathrm{vec}(\Delta\theta_k^l) = \mathrm{vec}(\boldsymbol{A}_l^{-1} \nabla_{\theta_k^l} \hat{\boldsymbol{g}}_k \boldsymbol{S}_l^{-1})$ 这里 $\boldsymbol{A}_l = \mathbb{E}[\boldsymbol{a}_l \boldsymbol{a}_l^{\mathrm{T}}]$, $S_l = \mathbb{E}[(\nabla_{s_l} \hat{g}_k)(\nabla_{\boldsymbol{s}_l} \hat{\boldsymbol{g}}_k)^{\mathrm{T}}]$（$\boldsymbol{A}_l, \boldsymbol{S}_l$ 通过计算片段的滚动平均值所得），$\boldsymbol{a}_l$ 是第 l 层的输入激活向量，$\boldsymbol{s}_l = \boldsymbol{W}_l \boldsymbol{a}_l$，$\mathrm{vec}(\cdot)$ 是把矩阵变换成一维向量的向量化变换

end for

由 K-FAC 近似自然梯度来更新策略：$\theta_{k+1} = \theta_k + \eta_k \Delta\theta_k$ 这里 $\eta_k = \min\left(\eta_{\max}, \sqrt{\frac{2\delta}{\boldsymbol{\theta}_k^{\mathrm{T}} \hat{\boldsymbol{H}}_k \boldsymbol{\theta}_k}}\right)$, $\hat{\boldsymbol{H}}_k^l = \boldsymbol{A}_l \otimes \boldsymbol{S}_l$

采用 Gauss-Newton 二阶梯度下降方法（并使用 K-FAC 近似）最小化均方误差来学习价值函数: $\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{\mathrm{T}} \left(V_\phi(S_t) - G_t\right)^2$

end for

B.4 高等深度强化学习

B.4.1 模仿学习

DAgger

算法 B.71 DAgger

初始化 $\mathcal{D} \leftarrow \emptyset$

初始化策略 $\hat{\pi}_1$ 为策略集 $\mathit{\Pi}$ 中任意策略

for i $= 1, 2, \cdots, N$ **do**

$\pi_i \leftarrow \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$

用 π_i 采样几个 T 步的轨迹

得到由 π_i 访问的策略和专家给出的动作组成的数据集 $\mathcal{D}_i = \{(s, \pi^*(s))\}$

聚合数据集：$\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$

在 $\mathcal{D}$ 上训练策略 $\hat{\pi}_{i+1}$

end for

返回策略 $\hat{\pi}_{N+1}$

B.4.2　基于模型的强化学习

Dyna-Q

算法 B.72 Dyna-Q

初始化 $Q(s,a)$ 和 Model(s,a)，其中 $s\in\mathcal{S}$，$a\in\mathcal{A}$

while(true):

(a) $s\leftarrow$ 当前（非终止）状态

(b) $a\leftarrow\epsilon$-greedy(s,Q)

(c) 执行决策行为 a; 观测奖励 r, 获得下一个状态 s'

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

(e) Model$(s,a)\leftarrow r,s'$

(f) 重复 n 次:

$s\leftarrow$ 随机历史观测状态

$a\leftarrow$ 在状态 s 下历史随机决策行为

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

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

朴素蒙特卡罗搜索（Simple Monte Carlo Search）

算法 B.73 朴素蒙特卡罗搜索

固定模型 $\mathcal{M}$ 和模拟策略 π

for 每个动作 $a\in\mathcal{A}$ **do**

for 每个片段 $k\in\{1,2,\cdots,K\}$ **do**

根据模型 $\mathcal{M}$ 和模拟策略 π, 从当前状态 S_t 开始在环境中展开

记录轨迹 $\{S_t,a,R_{t+1}^k,S_{t+1}^k,A_{t+1}^k,R_{t+2}^k,\cdots S_T^k\}$

计算从每个 S_t 开始的累积奖励 $G_t^k=\sum_{j=t+1}^{\mathrm{T}}R_j^k$

end for

$Q(S_t,a)=\frac{1}{K}\sum\limits_{k=1}^{K}G_t^k$

end for

返回当前最大 Q 值的动作 $A_t=\arg\max_{a\in\mathcal{A}}Q(S_t,a)$

蒙特卡罗树搜索（Monte Carlo Tree Search）

算法 B.74 蒙特卡罗树搜索

固定模型 $\mathcal{M}$
初始化模拟策略 π
for 每个动作 $a \in \mathcal{A}$ **do**
  for 每个片段 $k \in \{1, 2, \cdots, K\}$ **do**
    根据模型 $\mathcal{M}$ 和模拟策略 π 从当前状态 S_t 在环境中展开
    记录轨迹 $\{S_t, a, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, \cdots S_T\}$
    用从 (S_t, A_t)，$A_t = a$ 开始的平均回报更新每个 $(S_i, A_i), i = t, \cdots, T$ 的 Q 值
    由当前的 Q 值更新模拟策略 π
  end for
end for
返回当前最大 Q 值的动作 $A_t = \arg\max_{a \in \mathcal{A}} Q(S_t, a)$

Dyna-2

算法 B.75 Dyna-2

function LEARNING
  初始化 $\mathcal{F}_s$ 和 $\mathcal{F}_r$
  $\theta \leftarrow 0$ # 初始化长期存储空间中网络参数
  loop
    $s \leftarrow S_0$
    $\overline{\theta} \leftarrow 0$ # 初始化短期存储空间中网络参数
    $z \leftarrow 0$ # 初始化资格迹
    SEARCH(s)
    $a \leftarrow \pi(s; \overline{Q})$ # 基于和 $\overline{Q}$ 相关的策略选择决策动作
    while s 不是终结状态 **do**
      执行 a, 观测奖励 r 和下一个状态 s'
      $(\mathcal{F}_s, \mathcal{F}_r) \leftarrow$ UpdateModel(s, a, r, s')
      SEARCH(s')
      $a' \leftarrow \pi(s'; \overline{Q})$ # 选择决策动作使其用于下一个状态 s'
      $\delta \leftarrow r + Q(s', a') - Q(s, a)$ # 计算 TD-error
      $\theta \leftarrow \theta + \alpha(s, a)\delta z$ # 更新长期存储空间中网络参数
      $z \leftarrow \lambda z + \phi$ # 更新资格迹
      $s \leftarrow s', a \leftarrow a'$
    end while

end loop
end function

function SEARCH(s)
while 时间周期内 **do**
$\overline{z} \leftarrow 0$　　# 清除短期存储的资格迹
$a \leftarrow \overline{\pi}(s; \overline{Q})$　　# 基于和 $\overline{Q}$ 相关的策略决定决策动作
while s 不是终结状态 **do**
$s' \leftarrow \mathcal{F}_s(s, a)$　　# 获得下一个状态
$r \leftarrow \mathcal{F}_r(s, a)$　　# 获得奖励
$a' \leftarrow \overline{\pi}(s'; \overline{Q})$
$\overline{\delta} \leftarrow R + \overline{Q}(s', a') - \overline{Q}(s, a)$　　# 计算 TD-error
$\overline{\theta} \leftarrow \overline{\theta} + \overline{\alpha}(s, a)\overline{\delta}\overline{z}$　　# 更新短期存储空间中网络参数
$\overline{z} \leftarrow \overline{\lambda}\overline{z} + \overline{\phi}$　　# 更新短期存储的资格迹
$s \leftarrow s', a \leftarrow a'$
end while
end while
end function

B.4.3　分层强化学习

战略专注作家（STRategic Attentive Writer，STRAW）

算法 B.76 STRAW 中的计划更新

if $g_t = 1$ **then**
计算动作-计划的注意力参数 $\psi_t^A = f^{\psi}(z_t)$
应用专注阅读：$\beta_t = \text{read}(\boldsymbol{A}^{t-1}, \psi_t^A)$
计算中间表示 $\epsilon_t = h(\text{concat}(\beta_t, z_t))$
计算承诺-计划的注意力参数 $\psi_t^c = f^c(\text{concat}(\psi_t^A, \epsilon_t))$
更新 $\boldsymbol{A}^t = \rho(\boldsymbol{A}^{t-1}) + \text{write}(f^A(\epsilon_t), \psi_t^A)$
更新 $\boldsymbol{c}_t = \text{Sigmoid}(\boldsymbol{b} + \text{write}(e, \psi_t^c))$
else
更新 $\boldsymbol{A}^t = \rho(\boldsymbol{A}^{t-1})$
更新 $\boldsymbol{c}_t = \rho(\boldsymbol{c}_{t-1})$
end if

B.4.4 多智能体强化学习

多智能体 Q-Learning（Multi-Agent Q-Learning）

算法 B.77 多智能体一般性 Q-learning

设定 Q 表格中初始值 $Q_i(s, a_i, \boldsymbol{a}_{-i}) = 1, \forall i \in \{1, 2, \cdots, m\}$
for episode = 1 to M **do**
  设定初始状态 $s = S_0$
  for step = 1 to T **do**
    每个智能体 i 基于 $\pi_i(s)$ 选择决策行为 a_i , 其行为是根据当前 $\boldsymbol{Q}$ 中所有智能体混合纳什均衡决策策略
    观测经验 $(s, a_i, \boldsymbol{a}_{-i}, r_i, s')$ 并将其用于更新 Q_i
    更新状态 $s = s'$
  end for
end for

多智能体深度确定性策略梯度（Multi-Agent Deep Deterministic Policy Gradient，MADDPG）

算法 B.78 多智能体深度确定性策略梯度

for episode = 1 to M **do**
  设定初始状态 $s = S_0$
  for step = 1 to T **do**
    每个智能体 i 基于当前决策策略 π_{θ_i} 选择决策行为 a_i
    同时执行所有智能体的决策行为 $\boldsymbol{a} = (a_1, a_2, \cdots, a_m)$
    将 $(s, \boldsymbol{a}, r, s')$ 存在回放缓冲区 $\mathcal{M}$
    更新状态 $s = s'$
    for 智能体 i = 1 to m **do**
      从回访缓冲区 $\mathcal{M}$ 中采样批量历史经验数据
      对于行动者和批判者网络，计算网络参数梯度并根据梯度更新参数
    end for
  end for
end for

B.4.5　并行计算

异步优势 Actor-Critic（Asynchronous Advantage Actor-Critic，A3C）

算法 B.79 异步优势 Actor-Critic (Actor-Learner)

超参数: 总探索步数 $T_{\max}$，每个周期内最多探索步数 $t_{\max}$

初始化步数 $t=1$

while $T \leqslant T_{\max}$ **do**

　初始化网络参数梯度: $\mathrm{d}\theta=0$ 和 $\mathrm{d}\theta_v=0$

　和参数服务器保持同步并获得网络参数 $\theta'=\theta$ 和 $\theta'_v=\theta_v$

　$t_{\text{start}}=t$

　设定每个探索周期初始状态 S_t

　while 达到终结状态 or $t-t_{\text{start}}==t_{\max}$ **do**

　　基于决策策略 $\pi(S_t|\theta')$ 选择决策行为 a_t

　　在环境中采取决策行为，获得奖励 R_t 和下一个状态 S_{t+1}

　　$t=t+1, T=T+1$。

　end while

　if 达到终结状态 **then**

　　$R=0$

　else

　　$R=V(S_t|\theta'_v)$

　end if

　for $i=t-1,t-2,\cdots,t_{\text{start}}$ **do**

　　更新折扣化奖励 $R=R_i+\gamma R$

　　积累参数梯度 θ', $\mathrm{d}\theta=\mathrm{d}\theta+\nabla_{\theta'}\log\pi(S_i|\theta')(R-V(S_i|\theta'_v))$

　　积累参数梯度 θ'_v, $\mathrm{d}\theta_v=\mathrm{d}\theta_v+\partial(R-V(S_i|\theta'_v))^2/\partial\theta'_v$

　end for

　基于梯度 $\mathrm{d}\theta$ 和 $\mathrm{d}\theta_v$ 异步更新 θ 和 θ_v

end while

分布式近端策略优化（Distributed Proximal Policy Optimization，DPPO）

算法 B.80 DPPO (chief)

超参数: workers 数目 W, 可获得梯度的 worker 数目门限值 D, 次迭代数目 M, B

输入: 初始全局策略网络参数 θ, 初始全局价值网络参数 ϕ

for $\mathrm{k}=0,1,2,\cdots$ **do**

　for $m\in\{1,\cdots,M\}$ **do**

等待至少可获得 $W-D$ 个 worker 计算出来梯度 θ，去梯度的均值并更新全局梯度 θ

end for

for $b \in \{1, \cdots, B\}$ **do**

等待至少可获得 $W-D$ 个 worker 计算出来梯度 ϕ，去梯度的均值并更新全局梯度 ϕ

end for

end for

算法 B.81 DPPO (PPO-Penalty worker)

超参数: KL 惩罚系数 λ, 自适应参数 $a=1.5, b=2$, 次迭代数目 M, B

输入: 初始局部策略网络参数 θ, 初始局部价值网络参数 ϕ

for k $= 0, 1, 2, \cdots$ **do**

通过在环境中采用策略 π_θ 收集探索轨迹 $\mathcal{D}_k = \{\tau_i\}$

计算 rewards-to-go $\hat{G}_t$

基于当前价值函数 V_{ϕ_k} 计算对 advantage 的估计，$\hat{A}_t$（可选择使用任何一种 advantage 估计方法）

存储部分轨迹信息

$\pi_{\text{old}} \leftarrow \pi_\theta$

for $m \in \{1, \cdots, M\}$ **do**

$J_{\text{PPO}}(\theta) = \sum_{t=1}^{\mathrm{T}} \frac{\pi_\theta(A_t|S_t)}{\pi_{\text{old}}(A_t|S_t)} \hat{A}_t - \lambda \mathrm{KL}[\pi_{\text{old}}|\pi_\theta] - \xi \max(0, \mathrm{KL}[\pi_{\text{old}}|\pi_\theta] - 2\mathrm{KL}_{\text{target}})^2$

if $\mathrm{KL}[\pi_{\text{old}}|\pi_\theta] > 4\mathrm{KL}_{\text{target}}$ **then**

break 并继续开始 $k+1$ 次迭代

end if

计算 $\nabla_\theta J_{\text{PPO}}$

发送梯度数据 θ 到 chief

等待梯度被接受或被舍弃，更新网络参数

end for

for $b \in \{1, \cdots, B\}$ **do**

$L(\phi) = -\sum_{t=1}^{\mathrm{T}} (\hat{G}_t - V_\phi(S_t))^2$

计算 $\nabla_\phi L$

发送梯度数据 ϕ 到 chief

等待梯度被接受或被舍弃，更新网络参数

end for

计算 $d = \hat{\mathbb{E}}_t \left[\mathrm{KL}[\pi_{\text{old}}(\cdot|S_t), \pi_\theta(\cdot|S_t)]\right]$

if $d < d_{\text{target}}/a$ **then**

$\lambda \leftarrow \lambda/b$

else if $d > d_{\text{target}} \times a$ **then**

$\lambda \leftarrow \lambda \times b$

end if

end for

算法 B.82 DPPO (PPO-Clip worker)

超参数: clip 因子 ϵ, 次迭代数目 M, B

输入: 初始局部策略网络参数 θ, 初始局部价值网络参数 ϕ

for k $= 0, 1, 2, \cdots$ **do**

通过在环境中采用策略 π_θ 收集探索轨迹 $\mathcal{D}_k = \{\tau_i\}$

计算 rewards-to-go $\hat{G}_t$

基于当前价值函数 V_{ϕ_k} 计算对 advantage 的估计，$\hat{A}_t$（可选择使用任何一种 advantage 估计方法）

存储部分轨迹信息

$\pi_{\text{old}} \leftarrow \pi_\theta$

for $m \in \{1, \cdots, M\}$ **do**

通过最大化 PPO-Clip 目标更新策略:

$J_{\text{PPO}}(\theta) = \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{\mathrm{T}} \min\left(\frac{\pi_\theta(A_t|S_t)}{\pi_{\text{old}}(A_t|S_t)} \hat{A}_t(\frac{\pi(A_t|S_t)}{\pi_{\text{old}}(A_t|S_t)}, 1-\epsilon, 1+\epsilon)\hat{A}_t \right)$

计算 $\nabla_\theta J_{\text{PPO}}$

发送梯度数据 θ 到 chief

等待梯度被接受或被舍弃，更新网络参数

end for

for $b \in \{1, \cdots, B\}$ **do**

通过回归均方误差拟合价值方程:

$L(\phi) = -\frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{\mathrm{T}} \left(V_\phi(S_t) - \hat{G}_t \right)^2$

计算 $\nabla_\phi L$

发送梯度数据 ϕ 到 chief

等待梯度被接受或被舍弃，更新网络参数

end for

end for

Ape-X

算法 B.83 Ape-X (Actor)

超参数: 单次批量发送到回放缓冲区的数据大小 B，迭代数目 T

与学习者同步并获得最新的网络参数 θ_0

从环境中获得初始状态 S_0

for t $= 0, 1, 2, \cdots, T-1$ **do**
  基于决策策略 $\pi(S_t|\theta_t)$ 选择决策行为 A_t
  将经验 (S_t, A_t, R_t, S_{t+1}) 加入当地缓冲区
  if 当地缓冲区存储数据达到数目门限值 B **then**
    批量获得缓冲数据 B
    计算获得缓冲数据的优先级 p
    将批量缓冲数据和其更新的优先级发送回放缓冲区
  end if
  周期性同步并更新最新的网络参数 θ_t
end for

算法 B.84 Ape-X (Learner)

超参数: 学习周期数目 T
初始化网络参数 θ_0
for $t = 1, 2, 3, \cdots, T$ **do**
  从回放缓冲区中批量采样带有优先级的数据 (i, d)
  通过批数据进行模型训练
  更新网络参数 θ_t
  对于批数据 d 计算优先级 p
  更新回放缓冲区中索引 i 数据的优先级 p
  周期性地从回放缓冲区中删除低优先级的数据
end for