Chapter 1. Why Reinforcement Learning

强化学习

Reinforcement Learning (RL) 根据背景做出最佳决策。它通过尝试自己学习。它用于按次序决策，行动导致探索和环境的情况下表现最佳。其优势在于它优化长期多步的奖励。它广泛用于优化模型，无需生成随机样本离线拟合，因此十分迅速。

首个 RL 算法

Value estimate

$V(s,w) = w_0 s_0 + w_1 s_1 + \cdots + w_n s_n = w^T s$ ，$\vec{s}$ 为输入向量，二进制

Prediction error

$\delta = E - V(s,w)$ E为实际结果

Weight update rule

$w \leftarrow w - \alpha \delta x(s)$

Chapter 2. Markov Decision Processes, Dynamic Programming, and Monte Carlo Methods

Multi-Arm Bandit Testing

Policy Evaluation: 价值函数

行动 a 是集合 A 的元素，奖励 r 是 R 的元素，这些都是随机的

平均奖励计算：$r^{avg}(a) = \frac{1}{N(a)} \sum_{i=1}^{N(a)} r(a_i) = \frac{r_1 + \cdots + r_{N(a)}}{N(a)}$

Inline value function:

$$r_N^{avg} \leftarrow \frac{1}{N} \sum_{i=1}^{N} r_i \leftarrow \frac{1}{N}\left(r_N + \sum_{i=1}^{N-1} r_i\right) \leftarrow r_{N-1}^{avg} + \frac{1}{N}\left(r_N - r_{N-1}^{avg}\right)$$

更普遍来说，即 $r \leftarrow r + \alpha(r - r')$

Policy Improvement: 选最好的行动

ε-greedy bandit 算法

1. 输入一个探索概率 ε ($\varepsilon \in [0,1]$)
2. 初始化 $r^{avg}(a) \leftarrow 0$, $N(a) \leftarrow 0$, 对每个 $a \in A$
3. 循环:
4. $a \leftarrow \begin{cases} \underset{a \in A(s)}{\arg\max}\, r^{avg}(as) & \text{概率 } 1-\varepsilon \\ \text{随机 } a & \text{概率 } \varepsilon \end{cases}$
5. 执行 a, 获得奖励 r
6. $N(a) \leftarrow N(a)+1$
7. $r^{avg}(a) \leftarrow r^{avg}(a) + \frac{1}{N(a)}(r - r^{avg}(a))$

网络模拟算法

function Environment(a, p(a))

输入 a, 概率 p(a)

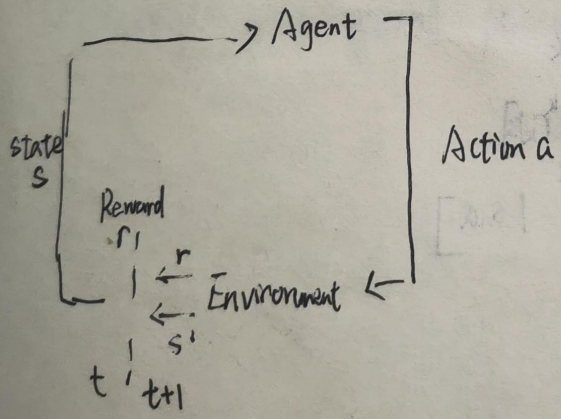
输出 $r \leftarrow 1$ (p(a)) 或 $r \leftarrow 0$ (1-p(a)), 以概率

改进 ε-greedy 算法

基于当前估计下的奖励分布选择行动更好，因此以概率选取行动代替直接选择行动，以期望奖励作概率区分，这里使用 softmax 函数

另一种是随时间减小 ε，称为 annealing；第三种探索未遇见的情况 (UCB)，通常不需要使用 ε

马尔可夫决策过程 (MDP)



Agent 影响离散系统，通过选择 action，目的为选择一系列行动最优化系统

其中 state 和 reward 为随机变量

MDP 转移模型，$p(s', r|s, a)$

实例：Inventory controll

$S=\{0,1,2\}$，库存容量　$A=\{$补货(restock)，不补(none)$\}$

$p(sale)$ 必须为 0.7

$$r=\begin{cases}1 & s>0 \text{ and a sale}\\ 1 & a=\text{restock and a sale}\\ 0\end{cases}$$

MDP 转移表

s	a	s'	p(s'|s,a)	r
0	none	0	1-p(sale)	0
0	none	0	p(sale)	0
0	re	1	1-p(sale)	0
0	re	0	p(sale)	1
1	no	1	1-p	0
1	no	0	p	1
1	re	2	1-p	0
1	re	1	p	1
2	no	2	1-p	0
2	no	1	p	1
2	re	2	p	1

转移矩阵

$$p(s',a=\text{none})=\begin{bmatrix}1 & 0 & 0\\ p & 1-p & 0\\ 0 & p & 1-p\end{bmatrix}$$

$$p(s',a=\text{restock})=\begin{bmatrix}p & 1-p & 0\\ 0 & p & 1-p\\ 0 & 0 & 1\end{bmatrix}$$

政策与价值函数

回报 $G=r+r'+\cdots+r_T$

折扣 Discounted return $G=r+\gamma r'+\gamma^2 r''+\cdots=\sum_{k=0}^{T}\gamma^k r_k$

State-Value 函数 $V_\pi(s)=E_\pi[G|s]=E_\pi[\sum_{k=0}^{T}\gamma^k r_k|s]$

Action-Value 函数 $Q_\pi(s,a)=E[G|s,a]=E_\pi[\quad|s,a]$

Optimal value 函数 $V_*(s)=\arg\max Q_{\pi_*}(s,a_s)$

蒙特卡罗政策生成

通过输入随机观察输出特征

On-policy MC 算法

1. input: a policy function, $\pi(a|s, q_\pi(s,a))$
2. Initialize $Q(s,a) \leftarrow 0$, for all $s \in \Phi$, $a \in A(s)$
3. loop:
4. Generate full episode trajectory following π
5. Initialize $G \leftarrow 0$
6. loop for each step of episode, $t = T-1, T-2 \cdots 0$
   $G \leftarrow \gamma G + r$
   if (s,a) not in $(s_0, a_0) \cdots (s_{t+1}, a_{t+1})$
   Append G to Returns (s,a)
   $Q(s,a) \leftarrow$ average (Returns (s,a))

动态规划下的 Value Iteration

1. input, 转移概率 $p(s', r|s,a)$, stopping threshold $\theta > 0$
2. 初始化 $V(s)$, for all $s \in \Phi$, $\nabla \leftarrow 0$
3. do:
4. loop for each $s \in \Phi$
   $v \leftarrow V(s)$
   $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s,a)[r + \gamma V(s')]$
   $\nabla \leftarrow \max(\nabla, |v - V(s)|)$

while $\nabla > \theta$

output an optimal policy, $\pi(s) = \arg\max_{a_s \in A(s)} \sum_{s',r} p(s', r|s, a_s)[r + \gamma V(s')]$

Chapter 3. Temporal Difference Learning, Q-Learning and n-Step algorithms.

Temporal-Difference (TD) Learning 的公式

Online 蒙特卡洛 state-value 函数

$$V_\pi(s) = E_\pi[G|s] \leftarrow E_\pi[V_\pi(s) + \alpha(G - V_\pi(s))|s]$$

Online 动态规划 state-value 函数

$$V_\pi(s) = E_\pi[G|s] \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$$

TD state-value 函数

$$V_\pi(s) = E_\pi[V_\pi(s) + \alpha(r + \gamma V_\pi(s') - V_\pi(s))|s]$$

Online TD state-value 函数

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha(r + \gamma V_\pi(s') - V_\pi(s))$$

Q-Learning

它是 TD 的一个补充法

Q-Learning online update rule to estimate the expected return

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \operatorname{argmax}_{a_s \in A(s)} Q(s',a_s) - Q(s,a))$$

Q-Learning (off-policy TD)

1. input: a policy that uses the action-value function. $\pi(a|s, Q(s,a))$
2. Initialize $Q(s,a) \leftarrow 0$, for all $s \in \mathcal{S}$, $a \in A(s)$

loop: for each episode

Initialize environment to provide s

do:

Choose a from s using π, breaking ties randomly

Take action a, and observe r, s'.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \operatorname{argmax}_{a_s \in A(s)} Q(s',a_s) - Q(s,a)]$$

$s \leftarrow s'$

while s is not terminal

SARSA

它提供更普遍的TD解法

SARSA online estimate of the action-value function

$Q(S,A) \leftarrow Q(S,A) + \alpha(r + \gamma Q(S',A') - Q(S,A))$

SARSA (on-policy TD)

1. Input: a policy that uses the action-value function, $\pi(a|s, Q(s,a))$
2. Initialize $Q(s,a) \leftarrow 0$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$
3. Loop: for episode

   Initialize environment to provide s

   do:

   Choose a from s using π, breaking ties randomly

   Take action, a, and observe r, s'

   Choose a' from s' using π, breaking ties randomly

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

   $s \leftarrow s'$, $a \leftarrow a'$

   while s is not terminal

~~Q-learning 与 SARSA~~

n-step 算法

2-step expected return

$G_{t:t+2} = r + \gamma r' + \gamma^2 Q(s'', a'')$

n-step expected return

$G_{t:t+n} = r + \gamma r' + \gamma^2 r'' + \cdots + \gamma^{n-1} r_{n-1} + \gamma^n Q(s_n, a_n)$

n-step TD update rule to estimate the expected return using the action-value function

$Q(s,a) \leftarrow Q(s,a) + \alpha(G_{t:t+n} - Q(s,a))$

no

n-step SARSA

1. input: a policy that uses the action-value function $\pi(a|s, Q(s,a))$
   step size $0<\alpha<1$, a positive integer n
2. Initialize $Q(s,a) \leftarrow 0$, for all $s \in \mathcal{S}$, $a \in A(s)$
3. loop for each episode:
   Initialize $T \leftarrow \infty$, $t \leftarrow 0$, replay buffers for S and A, and initialize states
   Chuose a from s using π, breaking ties randomly
   do:
      if $t<T$:
         Take action, a, and observe s', r
         if s' is terminal
            $T \leftarrow t+1$
         else:
            Chuse a' from s' using π, breaking ties randomly
      $\tau \leftarrow t-n+1$
      if $\tau+n<T$
         $G \leftarrow G+\gamma^n Q(S_{\tau+n}, a_{\tau+n})$
      $Q(S_\tau, a_\tau) \leftarrow Q(S_\tau, a_\tau)+\alpha[G-Q(S_\tau, a_\tau)]$
   $t \leftarrow \tau+1$. $s \leftarrow s'$, $a \leftarrow a'$
   while $\tau \neq T-1$

Eligibility Traces

SARSA(λ)

1. input: a policy that uses the action-value function, $\pi(a|s, Q(s,a))$.
   step size $0<\alpha<1$, a tracer decay rate $0 \leq \lambda \leq 1$
2. Initialize $Q(s,a) \leftarrow 0$, for all $s \in \varphi$, $a \in A(s)$
3. loop for each episode
   Initialize $z(s,a) \leftarrow 0$, for all $s \in \varphi$, $a \in A(s)$, and default states
   Choose a from s' using π, breaking ties randomly

do:
Take action a, and observe r, s'
Choose a' from s' using π, breaking ties randomly
$\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
$z(s, a) \leftarrow z(s, a) + 1$
for each $s \in \mathcal{S}$
for each $a \in A(s)$
$Q(s, a) \leftarrow Q(s, a) + \alpha \delta z(s, a)$
$z(s, a) \leftarrow \gamma \lambda z(s, a)$
$s \leftarrow s', a \leftarrow a'$
while s is not terminal

Chapter 4 Deep Q-Networks

Q-learning 或 SARSA 适用于小型、离散的状态，处理更大的最常用深度人工神经网络

Deep Q-Learning
对DQN的改进
Choosing an action with DQN

$$a \leftarrow \underset{a_s \in A(s)}{\operatorname{argmax}} Q(s, a_s; \theta)$$

DQN loss function

$$L(\theta) = E_\pi[(y - Q(s, a; \theta))^2]$$

Rainbow DQN
对DQN进行6项补充，其二为 double Q-Learning 和 n-step returns
最重要的是对 agents 重新引入概率

Distributional Bellman equation

$$Z(s, a) \overset{D}{\sim} R(s, a) + \gamma Z(s', A')$$

Chapter 5. Policy Gradient Methods

直接学习policy的益处

"Optimal Policies"定义policy为给定动作行动的概率. Q-Learning通过贪婪的动产生optimal policy来给出policy. 这一简化适用于deterministic problems

但采用随机policy可通过随机观测进行随机选取.

另外. 使用随机policy能干净地encapsulate the exploration dilemma

更长远的好处是它的action preferences随时间平滑变化

计算Policy的梯度

在此问题中. performance用$J(\theta)$表示. θ为模型参数

policy objective function

$J(\theta)=E_\pi[G|s,a]$

Vanilla gradient ascent

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Policy gradient 理论

Gradient of a policy

$$\begin{aligned}
\nabla J(\theta) &= \nabla E_\pi[G|s,a] \\
&= \nabla \sum_{a\in A}[Q(s,a)\pi(a|s)] \\
&\propto \sum_{a\in A}[Q(s,a)\nabla\pi(a|s)] \\
&\propto \sum_{a\in A}[Q(s,a)\pi(a|s)\cdot\frac{\nabla\pi(a|s)}{\pi(a|s)}] \\
&\propto \sum_{a\in A}[Q(s,a)\nabla\pi(a|s)\nabla\ln\pi(a|s)] \\
&\propto E_\pi[G\nabla\ln\pi(a|s)]
\end{aligned}$$

Policy gradient update rule

$\theta \leftarrow \theta + \alpha G \nabla \ln\pi(a|s,\theta)$

Policy functions

* 线性 policy

Logistic function : $f(x) = \frac{1}{1+e^{-x}}$

Logistic policy

$$\pi_\theta(a|s) = \begin{pmatrix} \pi_\theta(a=0|s) \\ \pi_\theta(a=1|s) \end{pmatrix} = \begin{cases} \frac{1}{1+e^{-\theta_0^T s}} \\ 1 - \frac{1}{1+e^{-\theta_1^T s}} \end{cases}$$

Logistic policy gradient

$$\nabla \ln \pi(a|s,\theta) = \begin{bmatrix} s - s\pi(\theta_0^T s) \\ -s\pi(\theta_1^T s) \end{bmatrix}$$

Softmax policy

$$\pi_\theta(a|s) = \frac{e^{\theta^T s}}{\sum_a e^{\theta_a^T s}}$$

Softmax policy gradient

$$\nabla \ln \pi(a|s,\theta) = s - \sum_a s\pi(\theta^T s)$$

Basic Implementations

蒙特卡洛 (Reinforce)

REINFORCE 算法

1. input: a differentiable parameterized policy $\pi(a|s,\theta)$, step size $\alpha > 0$
2. initialize θ arbitrarily
3. loop for each episode

   generate trajectory following $\pi(\cdot|\cdot,\theta)$

   for each step in episode $t = 0, 1, \cdots, T$

   $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$

   $\theta \leftarrow \theta + \alpha \gamma^t G \nabla \pi(a|s,\theta)$

   end for

Reinforce with Baseline

REINFORCE with baseline algorithm

1. input: a differentiable parameterized policy $\pi(a|s,\theta)$
   a differentiable parameterized state-value function $V(s,w)$
   step size $0<\alpha_w<0$ and $0<\alpha_\theta<1$
2. Initialize θ and w arbitrarily
3. loop for each episode:
   Generate trajectory following $\pi(\cdot|\cdot,\theta)$:
   for each step in episode $t\leftarrow 0,1,\cdots,T$:
   $G\leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
   $\delta\leftarrow G-V(s,w)$
   $w\leftarrow w+\alpha_w \delta \nabla V(s,a_w)$
   $\theta\leftarrow \theta+\alpha_\theta \gamma^t \delta \nabla \pi(a_t|s,\theta)$
   end for

n-Step Actor-Critic and Advantage Actor-Critic (A2C)

n-step actor-critic

1. input: a differentiable parameterized policy $\pi(a|s,\theta)$
   a differentiable parameterized state-value function $V(s,w)$
   Step size α_w and $\alpha_\theta>0$
2. Initialize θ and w
3. loop for each episode
   Initialize $I\leftarrow 1$, $t\leftarrow 0$, and s
   while s is not terminal
   Choose a from s using π, breaking ties randomly
   Take action a, observe s', r
   $\delta\leftarrow G_{t:t+n}-V(s,\theta)$
   $w\leftarrow w+\alpha_w \delta \nabla V(s,w)$
   $\theta\leftarrow \theta+\alpha_\theta I \delta \nabla \ln \pi(a|s,\theta)$
   $I\leftarrow \gamma I$
   $s\leftarrow s'$

Advantage function approximation for policy gradient algorithms

$$A^{\pi}(s,a) = Q(s,a) - V(s)$$
$$= r + Q(s',a') - V(s)$$
$$= r + V_{\pi}(s') - V(s)$$

Eligibility Traces Actor-Critic

Actor-Critic with eligibility traces

1. input: a differentiable parameterized policy $\pi(a|s,\theta)$, a differentiable parameterized state-value function $V(s,w)$ step size $\gamma < \alpha^{w}, \alpha^{\theta} < 1$, and trace-decay rates $0 \le \lambda_{\theta}, \lambda_{w} \le 1$

2. initialize θ and w arbitrarily

3. loop for each episode

Initialize $t \leftarrow 0$. $I \leftarrow 1$ and s

$z_w \leftarrow 0$. (same dimensionality as w) and

$z_{\theta} \leftarrow 0$. (same dimensionality as θ)

while s is not terminal:

choose a from s using π, breaking ties randomly

Take action a, observe s', r

$\delta \leftarrow r + \gamma V(s',w) - V(s,\theta)$

$z_w \leftarrow \gamma \lambda_w z_w + \nabla V(s,w)$

$z_{\theta} \leftarrow \gamma \lambda_{\theta} z_{\theta} + I \nabla \ln \pi(a|s,\theta)$

$w \leftarrow w + \alpha_w \delta z_w$

$\theta \leftarrow \theta + \alpha_{\theta} I \delta z_{\theta}$

$I \leftarrow \gamma I$

$s \leftarrow s'$

Equivalent implements of the policy gradient algorithm

$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \ln \pi_\theta(a|s) \chi_t]$ REINFORCE

$= E_{\pi_\theta}[\nabla_\theta \ln \pi_\theta(a|s)(G_t - V_w(s))]$ REINFORCE with Baseline

$= E_{\pi_\theta}[\nabla_\theta \ln \pi_\theta(a|s) Q_w(s,a)]$ Actor-Critic

$= E_{\pi_\theta}[\nabla_\theta \ln \pi_\theta(a|s) A_w(s,a)]$ Advantage Actor-Critic

$= E_{\pi_\theta}[\nabla_\theta \ln \pi_\theta(a|s) \delta]$ Temporal-difference Actor-Critic

$= E_{\pi_\theta}[\nabla_\theta \ln \pi_\theta(a|s) \delta e]$ Eligibility Actor-Critic

Chapter 6. Beyond Policy Gradients

Off policy Algorithms

Importance sampling

$E_f[x] \doteq \sum_x x f(x)$

$E_g[x] \doteq \sum_x x g(x)$

$= \sum_x \frac{x g(x)}{f(x)} \cdot f(x)$

$= E_f[\frac{x g(x)}{f(x)}]$

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

Parameterized linear TD error

$\delta \doteq r + \gamma \theta^T \phi(s') - \theta^T \phi(s)$

GTD(0)算法

1. input: a policy that uses the parameterized action-value function. $\pi(a|s,a;\theta)$. learning rate parameters. $0 < \alpha < 1$, $0 < \beta < 1$
2. initialize θ and u arbitrarily
3. loop: for each episode

Initialize s

while s is not terminal:
  Choose a from s using π, breaking ties randomly
  Take action a, and observe r, s'
  $\delta \leftarrow r + \gamma \theta^T \phi(s') - \theta^T \phi(s)$
  $u \leftarrow u + \beta(\phi(s) - u)$
  $\theta \leftarrow \theta + \alpha(\phi(s) - \gamma \phi(s'))\phi(s)^T u$
  $s \leftarrow s'$

Greedy GQ

Greedy-GQ algorithm

....

q: $w \leftarrow w + \beta[\phi(s') - \phi(s)^T w]\phi(s)$

$\theta \leftarrow \theta + \alpha[\delta\phi(s) - \gamma(w^T\phi(s))\phi(s')]$

$s \leftarrow s'$

Off-policy Actor-Critics

Off-policy gradient estimate

$$\nabla_\theta J_\beta(\pi_\theta) = E_{s\sim\rho_\beta,\, a\sim\beta}\left[\frac{\pi_\theta(a|s)}{\beta_\theta(a|s)} \nabla_\theta \ln \pi_\theta(a|s) Q_\pi(s,a)\right]$$

Deterministic Policy Gradients

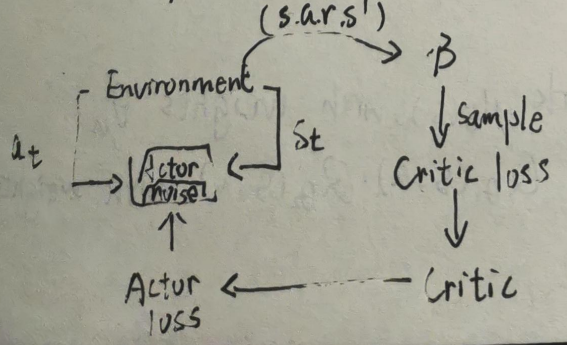
Off-policy vanilla and deterministic actor-critic objective function gradients, for comparison

$$\nabla_\theta J_\beta(\pi_\theta) = E_{s\sim\rho_\beta,\, a\sim\beta}\left[\frac{\pi_\theta(a|s)}{\beta_\theta(a|s)} \nabla_\theta \ln \pi_\theta(a|s) Q_\pi(s,a)\right]$$

$$\nabla_\theta J_\beta(\mu_\theta) = E_{s\sim\rho_\beta}\left[\nabla_\theta Q_\mu(s, \mu_\theta(s))\right]$$

$$= E_{s\sim\rho_\beta}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q_\mu(s,a)\right.$$

DDPG:



DDPG algorithm

1. input: an actor neural network, $\mu_\theta(s)$, with weights θ_μ,
   a critic neural network, $Q_{\theta_Q}(s,a)$, with weights θ_Q
   a replay buffer B
   the size of a training minibatch, N.
   the network update rate, τ,
   the discount-rate parameter, γ.
   and a noise function for exploration, $\mathcal{N}$
2. Initialize θ_μ, θ_Q, R.
   and the target weights, $\theta'_\mu \leftarrow \theta_\mu$, $\theta'_Q \leftarrow \theta_Q$
3. loop: for each episode
   Initialize s
   while s is not terminal:
   $a \leftarrow \mu_{\theta_\mu}(s) + \mathcal{N}$
   Take action, a, and observe r, s'
   Store (s, a, r, s') in B
   Sample a random minibatch of N transitions. (s_i, a_i, r_i, s_i') from B
   $Y_i \leftarrow r_i + \gamma Q_{\theta'_Q}(s_i', \mu_{\theta'_\mu}(s_i'))$
   Update the critic by minimizing: $\frac{1}{N}\sum_i (y_i - Q_{\theta_Q}(s_i, a_i))^2$
   Update the actor by minimizing: $\frac{1}{N}\sum_i (\nabla_{\theta_\mu} Q_{\theta_Q}(s_i, \mu_{\theta_\mu}(s_i)))^2$
   $\theta'_Q \leftarrow \tau\theta_Q + (1-\tau)\theta'_Q$
   $\theta'_\mu \leftarrow \tau\theta_\mu + (1-\tau)\theta'_\mu$
   $s \leftarrow s'$

TD3 algorithm

1. input an actor neural network, $\mu_\theta(s)$, with weights θ_μ
   two critic neural networks, $Q_{\theta_1}(s,a)$, $Q_{\theta_2}(s,a)$, with weights θ_1, θ_2

a relay buffer B,
the size of a training minibatch N
the network update rate τ
the discount-rate parameter γ.
a noise clipping threshold c
and a noise function for exploration $\mathcal{N}$

2. Initialize $\theta_\mu, \theta_1, \theta_2$.
and the target weights: $\theta'_\mu \leftarrow \theta_\mu, \theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2$

3. loop: for each episode
Initialize s
while s is not terminal
$a \leftarrow \mu_{\theta_\mu}(s) + \mathcal{N}$
Take action a, and observe r, s'
Store (s, a, r, s') in B
Sample a random minibatch of N transitions (s_i, a_i, r_i, s'_i) from B
$\tilde{a}_i \leftarrow \mu_{\theta'_\mu}(s'_i) + \text{clip}(\mathcal{N}, -c, c)$
$y_i \leftarrow r_i + \gamma \min_{j=1,2} Q_{\theta'_j}(s'_i, \tilde{a}_i)$
Update both critics, $\theta_{j \leftarrow 1,2}$ by minimizing: $\frac{1}{N}\sum_i (y_i - Q_{\theta_j}(s_i, a_i))^2$
if $t \bmod d$ then:
Update the actor by minimizing: $\frac{1}{N}\sum_i (\nabla_{\theta_\mu} Q_{\theta_1}(s_i, \mu_{\theta_\mu}(s_i)))^2$
Update both critics. $\theta_{j \leftarrow 1,2}$: $\theta'_j \leftarrow \tau\theta_j + (1-\tau)\theta'_j$
$\theta'_\mu \leftarrow \tau\theta_\mu + (1-\tau)\theta'_\mu$
end if
$s \leftarrow s'$

Trust Region Methods

Advantage actor-critic policy gradient

$L^{PG}(\theta) = E_{\pi_\theta}[\ln \pi_\theta(a|s) A_w(s,a)]$

Kullback-Leibler divergence

$$D_{KL}(P\|Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Reformulated advantage actor-critic loss function

$$\nabla J(\theta) = E_{\pi_\theta}[\nabla \ln \pi_\theta(a|s) A_w(s,a)]$$

$$= E_{\pi_\theta}\left[\frac{\nabla \pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_w(s,a)\right]$$

Presenting A2C in terms of maximization of the objective

$$\underset{\theta}{\text{maximize}}\ L^{PG}(\theta) = E_{\pi_\theta}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_w(s,a)\right]$$

Natural policy gradients optimization loss function

$$\underset{\theta}{\text{maximize}}\ L^{TRPO}(\theta) = E_{\pi_\theta}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_w(s,a)\right]$$

$$\text{subject to } E_{\pi_\theta}[KL[\pi_{\theta_{old}}(\cdot|s) \| \pi_\theta(\cdot|s)]] \leq \delta$$

Penalizing large changes in policy

$$\underset{\theta}{\text{maximize}}\ E_{\pi_\theta}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_w(s,a) - \beta KL[\pi_{\theta_{old}}(\cdot|s) \| \pi_\theta(\cdot|s)]\right]$$

Clipping large changes in policy according to the importance sampling ratio

$$L^{CLIP}(\theta) = E[\min(r(\theta)A(s,a), \text{clip}(r(\theta), 1-\varepsilon, 1+\varepsilon)\hat{a}(s,a)]$$

The final objective function for PPO

$$L^{CLIP+VF+S}(\theta) = E[L^{CLIP}(\theta) + C_1 L^{VF}(\theta) + C_2 H[\pi_\theta](s)]$$

Asynchronous PPO algorithm

1. for iteration = 0, 1, ...

   for agent = 0, 1, ..., N-1

   Run policy $\pi_{\theta_{old}}$ in environment for T time steps

   Compute advantage estimates $A_0, \ldots, A_{T-1}$

   Optimize L with respect to θ, for K epochs and minibatch size $M \leq NT$

   $\theta_{old} \leftarrow \theta$

Chapter 7 Learning All possible Polices with Entropy Methods

熵 Shannon entropy 以下简称 entropy，它衡量随机变量的信息量

随机变量的信息量

$$H(X) = E[I(X)] = -\sum_{x \in X} p(x) \log_b p(x)$$

Maximum Entropy Reinforcement Learning

Maximum entropy objective function

$$\pi^* = \arg\max_{\pi} \sum_{s \in \mathcal{S}, a \in \mathcal{A}} E(r + \alpha H(\pi(a|s)))$$

Soft Actor-Critic

Maximum entropy state-value function

$$V(s) = E[Q(s,a) + \alpha H(\pi(a|s))]$$

$$= E[Q(s,a) - \alpha \log \pi(a|s)]$$

Maximum entropy action-value function

$$Q(s,a) = E[r + \gamma E[V(s')]]$$

Maximum entropy action-value objective function

$$J_Q(\theta) = E[\frac{1}{2}(Q_\theta(s,a) - Q(s,a))^2]$$

$$= E[\frac{1}{2}(Q_\theta(s,a) - (r + \gamma E[V_\theta(s')]))^2]$$

SAC policy objective function

$$J_\pi(\phi) = E[\alpha \log(\pi_\phi(a|s)) - Q_\theta(s,a)]$$

SAC temperature parameter objective function

$$J_\alpha = E[-\alpha \log(\pi(a|s)) - \alpha \bar{H}]$$

Extentions to Maximum Entropy Methods

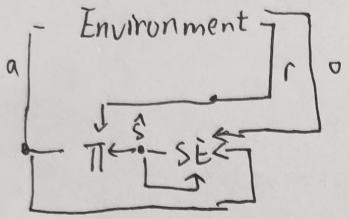
Deep soft policy gradient

$$\nabla J_\pi(\phi) = E[(Q_\theta(s,a) - \log \pi(a|s) - 1) \nabla_\phi \log \pi(a|s)]$$

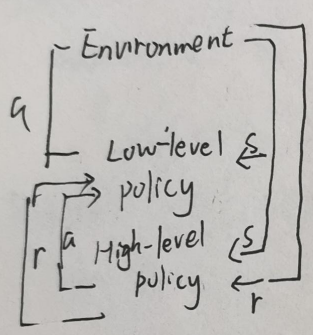
Chapter 8 Improving How an Agent Learns

Rethinking the MDP

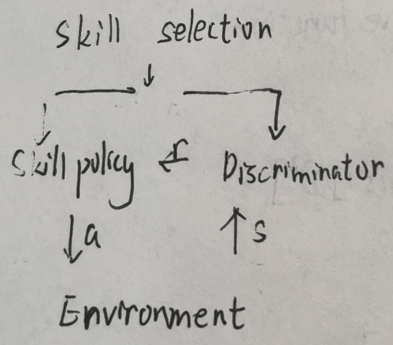
decompose a POMDP into an MDP and a state estimator



High-Low Hierarchies with Intrinsic Rewards (HIRO)

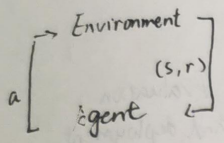


Schematic diagram of unsupervised skill learning framework (DIAYN)

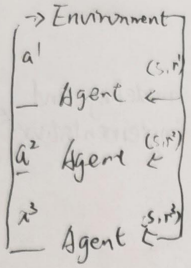


Schematic diagrams for different MARL frameworks.

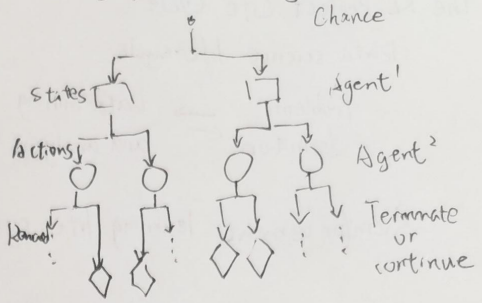
(a) MDP



(b) Markov game

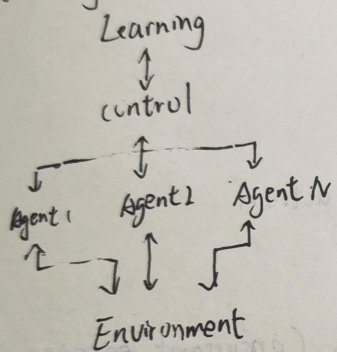


(c) Extensive-form game

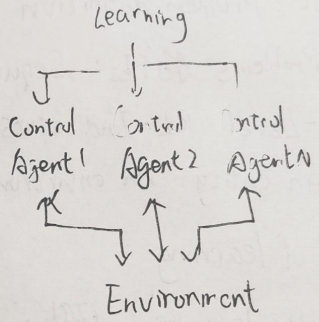


Centralized or Decentralized

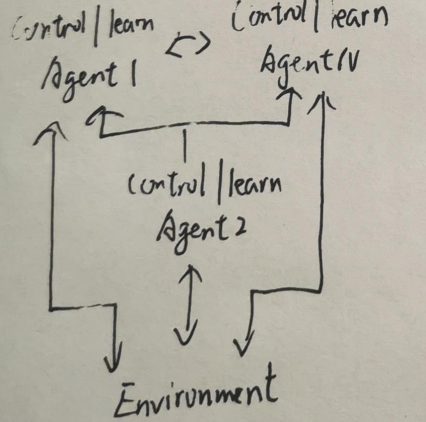
A) Fully centralized



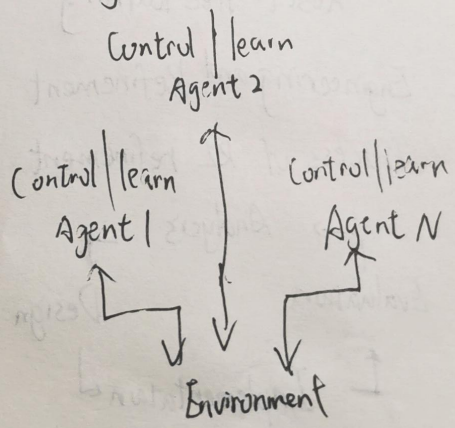
B) Centralized learning decentralized execution



C Decentralized with sharing



D) Fully decentralized



Chapter 9. Practical Reinforcement Learning

The RL Project Life Cycle

Data science life cycle

Problem definition ⇄ Data mining and analysis ⇄ modeling and implementation ⇄ Evaluation and deployment

Reinforcement learning life cycle

same ⇄ Environment and MDP refinement ⇄ same ⇄ same

RL Project: Problem definition

RL Problems 特征: Sequential, Strategic

Low-Level RL Indicators:

An entity. An environment. A state. An action

Types of learning:

Online learning. Offline or batch learning. Concurrent learning

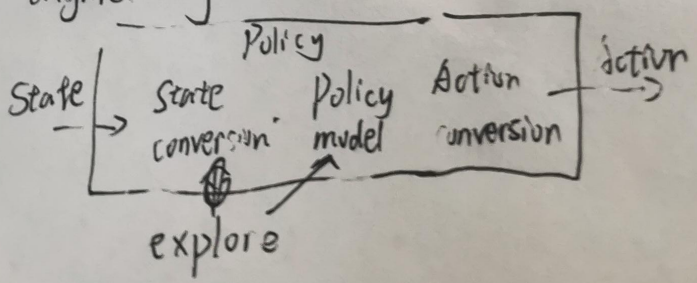
Reset-free learning

RL Engineering and Refinement

Process of RL refinement

Analysis → Design → Implementation → Evaluation → Analysis

Policy engineering



Chapter 10 Operational Reinforcement Learning

Implementation

Frameworks. Scaling RL. Evaluation

Deployment

Goals, Architecture, Ancillary Tooling Safety, Security and Ethics