目录

1 Template

算法[①]

1: Test

[①]this is footnote

2 Q learning算法

Q-learning算法[1]

初始化Q表$Q(s,a)$为任意值，但其中$Q(s_{terminal},)=0$，即终止状态对应的Q值为0
for 回合数 $=1,M$ **do**
  重置环境，获得初始状态s_1
  for 时步 $=1,T$ **do**
    根据$\varepsilon-greedy$策略采样动作a_t
    环境根据a_t反馈奖励r_t和下一个状态s_{t+1}
    更新策略：
    $Q(s_t,a_t)\leftarrow Q(s_t,a_t)+\alpha[r_t+\gamma\max_a Q(s_{t+1},a)-Q(s_t,a_t)]$
    更新状态$s_{t+1}\leftarrow s_t$
  end for
end for

[1]Reinforcement Learning: An Introduction

3 Sarsa算法

Sarsa算法[1]

初始化Q表$Q(s,a)$为任意值，但其中$Q(s_{terminal},)=0$，即终止状态对应的Q值为0
for 回合数 $=1,M$ **do**
  重置环境，获得初始状态s_1
  根据$\varepsilon-greedy$策略采样初始动作a_1
  for 时步 $=1,t$ **do**
    环境根据a_t反馈奖励r_t和下一个状态s_{t+1}
    根据$\varepsilon-greedy$策略s_{t+1}和采样动作a_{t+1}
    更新策略：
    $Q(s_t,a_t)\leftarrow Q(s_t,a_t)+\alpha[r_t+\gamma Q(s_{t+1},a_{t+1})-Q(s_t,a_t)]$
    更新状态$s_{t+1}\leftarrow s_t$
    更新动作$a_{t+1}\leftarrow a_t$
  end for
end for

[1] Reinforcement Learning: An Introduction

4 DQN算法

DQN算法[1]

初始化策略网络参数θ
复制参数到目标网络$\hat{Q} \leftarrow Q$
初始化经验回放D
for 回合数 $= 1, M$ **do**
重置环境，获得初始状态s_t
for 时步 $= 1, t$ **do**
根据$\varepsilon - greedy$策略采样动作a_t
环境根据a_t反馈奖励r_t和下一个状态s_{t+1}
存储transition即(s_t, a_t, r_t, s_{t+1})到经验回放D中
更新环境状态$s_{t+1} \leftarrow s_t$
更新策略：
从D中采样一个batch的transition
计算实际的Q值，即y_j[2]
对损失 $L(\theta) = (y_i - Q(s_i, a_i; \theta))^2$关于参数$\theta$做随机梯度下降[3]
end for
每C个回合复制参数$\hat{Q} \leftarrow Q$[4]]
end for

---

[1] Playing Atari with Deep Reinforcement Learning

[2] $y_i = \begin{cases} r_i & \text{对于终止状态} s_{i+1} \\ r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta) & \text{对于非终止状态} s_{i+1} \end{cases}$

[3] $\theta_i \leftarrow \theta_i - \lambda \nabla_{\theta_i} L_i(\theta_i)$

[4] 此处也可像原论文中放到小循环中改成每C步，但没有每C个回合稳定

5 PER-DQN算法

PER_DQN算法[1]

初始化策略网络参数θ
复制参数到目标网络$\hat{Q} \leftarrow Q$
初始化经验回放D
for 回合数 $= 1, M$ **do**
  重置环境，获得初始状态s_t
  for 时步 $= 1, t$ **do**
    根据$\varepsilon - greedy$策略采样动作a_t
    环境根据a_t反馈奖励r_t和下一个状态s_{t+1}
    存储transition即(s_t, a_t, r_t, s_{t+1})到经验回放D，并根据TD-error损失确定其优先级p_t
    更新环境状态$s_{t+1} \leftarrow s_t$
    更新策略：
    按照经验回放中的优先级别，每个样本采样概率为$P(j) = p_j^\alpha / \sum_i p_i^\alpha$，从$D$中采样一个大小为batch的transition
    计算各个样本重要性采样权重 $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
    计算TD-error δ_j ; 并根据TD-error更新优先级p_j
    计算实际的Q值，即y_j[2]
    根据重要性采样权重调整损失 $L(\theta) = (y_j - Q(s_j, a_j; \theta) \cdot w_j)^2$，并将其关于参数$\theta$做随机梯度下降[3]
  end for
  每C个回合复制参数$\hat{Q} \leftarrow Q$[4]]
end for

[1] Playing Atari with Deep Reinforcement Learning

[2] $y_i = \begin{cases} r_i & \text{对于终止状态} s_{i+1} \\ r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta) & \text{对于非终止状态} s_{i+1} \end{cases}$

[3] $\theta_i \leftarrow \theta_i - \lambda \nabla_{\theta_i} L_i(\theta_i)$

[4] 此处也可像原论文中放到小循环中改成每C步，但没有每C个回合稳定

6 NoisyDQN算法

NoisyDQN算法[1]

初始化策略网络每个参数（权重和偏置）对应的噪声变量μ和σ
复制参数到目标网络$\hat{Q} \leftarrow Q$
初始化经验回放D
for 回合数 $= 1, M$ **do**
  重置环境，获得初始状态s_t
  for 时步 $= 1, t$ **do**
    根据$\varepsilon - greedy$策略采样动作a_t
    环境根据a_t反馈奖励r_t和下一个状态s_{t+1}
    存储transition即(s_t, a_t, r_t, s_{t+1})到经验回放D中
    更新环境状态$s_{t+1} \leftarrow s_t$
    更新策略：
    从D中采样一个batch的transition
    计算实际的Q值，即y_j[2]
    对损失 $L(\mu, \sigma) = (y_i - Q(s_i, a_i; \mu, \sigma))^2$关于参数$\mu, \sigma$做随机梯度下降
    对目标网络和策略网络中的噪声项ϵ进行重置
  end for
  每C个回合复制参数$\hat{Q} \leftarrow Q$[3]
end for

[1] Playing Atari with Deep Reinforcement Learning

[2] $y_i = \begin{cases} r_i & \text{对于终止状态} s_{i+1} \\ r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta) & \text{对于非终止状态} s_{i+1} \end{cases}$

[3] 此处也可像原论文中放到小循环中改成每C步，但没有每C个回合稳定

7 Policy Gradient算法

REINFORCE算法：Monte-Carlo Policy Gradient[①]

初始化策略参数$\boldsymbol{\theta} \in \mathbb{R}^{d'}$(e.g., to $\mathbf{0}$)
for 回合数 $= 1, M$ **do**
  根据策略$\pi(\cdot \mid \cdot, \boldsymbol{\theta})$采样一个(或几个)回合的transition
  for 时步 $= 0, 1, 2, ..., T-1$ **do**
    计算回报$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    更新策略$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi \left(A_t \mid S_t, \boldsymbol{\theta}\right)$
  end for
end for

[①]Reinforcement Learning: An Introduction

8 Advantage Actor Critic算法

Q Actor Critic算法

```
初始化Actor参数θ和Critic参数w
for 回合数 = 1, M do
    根据策略π_θ(a|s)采样一个(或几个)回合的transition
    更新Critic参数[1]
    for 时步 = t+1, 1 do
        计算Advantage，即δ_t = r_t + γQ_w(s_{t+1}, a_{t+1}) − Q_w(s_t, a_t)
        w ← w + α_w δ_t ∇_w Q_w(s_t, a_t)
        a_t ← a_{t+1}, s_t ← s_{t+1}
    end for
    更新Actor参数θ ← θ + α_θ Q_w(s, a) ∇_θ log π_θ(a | s)
end for
```

[1] 这里结合TD error的特性按照从$t+1$到1计算法Advantage更方便

9 PPO-Clip算法

PPO-Clip算法[1][2]

初始化策略网络(Actor)参数θ和价值网络(Critic)参数ϕ
初始化Clip参数ϵ
初始化epoch数量K
初始化经验回放D
初始化总时步数$c = 0$
for 回合数 $= 1, 2, \cdots, M$ **do**
  重置环境，获得初始状态s_0
  for 时步 $t = 1, 2, \cdots, T$ **do**
    计数总时步$c \leftarrow c + 1$
    根据策略π_θ选择a_t
    环境根据a_t反馈奖励r_t和下一个状态s_{t+1}
    存储(s_t, a_t, r_t, s_{t+1})到经验回放D中
    if c被C整除[3] **then**
      for $k = 1, 2, \cdots, K$ **do**
        测试
      end for
      清空经验回放D
    end if
  end for
end for

[1] Proximal Policy Optimization Algorithms

[2] https://spinningup.openai.com/en/latest/algorithms/ppo.html

[3] 即每C个时步更新策略

10 DDPG算法

DDPG算法[1]

初始化critic网络$Q\left(s, a \mid \theta^{Q}\right)$和actor网络$\mu(s|\theta^{\mu})$的参数$\theta^{Q}$和$\theta^{\mu}$
初始化对应的目标网络参数，即$\theta^{Q'} \leftarrow \theta^{Q}, \theta^{\mu'} \leftarrow \theta^{\mu}$
初始化经验回放R
for 回合数 $= 1, M$ **do**
  选择动作$a_t = \mu\left(s_t \mid \theta^{\mu}\right) + \mathcal{N}_t$，$\mathcal{N}_t$为探索噪声
  环境根据a_t反馈奖励s_t和下一个状态s_{t+1}
  存储transition(s_t, a_t, r_t, s_{t+1})到经验回放R中
  更新环境状态$s_{t+1} \leftarrow s_t$
  更新策略：
  从R中取出一个随机批量的(s_i, a_i, r_i, s_{i+1})
  求得$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right)$
  更新critic参数，其损失为：$L = \frac{1}{N} \sum_i \left(y_i - Q\left(s_i, a_i \mid \theta^{Q}\right)\right)^2$
  更新actor参数：$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q\left(s, a \mid \theta^{Q}\right)\Big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu\left(s \mid \theta^{\mu}\right)\Big|_{s_i}$
  软更新目标网络：$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$， $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
end for

[1] Continuous control with deep reinforcement learning

11 SoftQ算法

SoftQ算法

初始化参数θ和ϕ
复制参数$\bar{\theta} \leftarrow \theta, \bar{\phi} \leftarrow \phi$
初始化经验回放D
for 回合数 $= 1, M$ **do**
  for 时步 $= 1, t$ **do**
    根据$\mathbf{a}_t \leftarrow f^{\phi}(\xi; \mathbf{s}_t)$采样动作，其中$\xi \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    环境根据a_t反馈奖励s_t和下一个状态s_{t+1}
    存储transition即(s_t, a_t, r_t, s_{t+1})到经验回放D中
    更新环境状态$s_{t+1} \leftarrow s_t$
    更新soft Q函数参数：
    对于每个$s_{t+1}^{(i)}$采样$\left\{\mathbf{a}^{(i,j)}\right\}_{j=0}^{M} \sim q_{\mathbf{a}'}$
    计算empirical soft values $V_{\text{soft}}^{\theta}(\mathbf{s}_t)$[1]
    计算empirical gradient $J_Q(\theta)$[2]
    根据$J_Q(\theta)$使用ADAM更新参数θ
    更新策略：
    对于每个$s_t^{(i)}$采样$\left\{\xi^{(i,j)}\right\}_{j=0}^{M} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    计算$\mathbf{a}_t^{(i,j)} = f^{\phi}\left(\xi^{(i,j)}, \mathbf{s}_t^{(i)}\right)$
    使用经验估计计算$\Delta f^{\phi}(\cdot; \mathbf{s}_t)$[3]
    计算经验估计$\frac{\partial J_\pi(\phi; \mathbf{s}_t)}{\partial \phi} \propto \mathbb{E}_{\xi}\left[\Delta f^{\phi}(\xi; \mathbf{s}_t) \frac{\partial f^{\phi}(\xi; \mathbf{s}_t)}{\partial \phi}\right]$，即$\hat{\nabla}_{\phi} J_{\pi}$
    根据$\hat{\nabla}_{\phi} J_{\pi}$使用ADAM更新参数$\phi$

  end for
  每C个回合复制参数$\bar{\theta} \leftarrow \theta, \bar{\phi} \leftarrow \phi$
end for

[1] $V_{\text{soft}}^{\theta}(\mathbf{s}_t) = \alpha \log \mathbb{E}_{q_{\mathbf{a}'}}\left[\frac{\exp\left(\frac{1}{\alpha} Q_{\text{soft}}^{\theta}(\mathbf{s}_t, \mathbf{a}')\right)}{q_{\mathbf{a}'}(\mathbf{a}')}\right]$

[2] $J_Q(\theta) = \mathbb{E}_{\mathbf{s}_t \sim q_{\mathbf{s}_t}, \mathbf{a}_t \sim q_{\mathbf{a}_t}}\left[\frac{1}{2}\left(\hat{Q}_{\text{soft}}^{\bar{\theta}}(\mathbf{s}_t, \mathbf{a}_t) - Q_{\text{soft}}^{\theta}(\mathbf{s}_t, \mathbf{a}_t)\right)^2\right]$

[3]
$$\begin{aligned}\Delta f^{\phi}(\cdot; \mathbf{s}_t) = & \mathbb{E}_{\mathbf{a}_t \sim \pi^{\phi}}\left[\kappa\left(\mathbf{a}_t, f^{\phi}(\cdot; \mathbf{s}_t)\right) \nabla_{\mathbf{a}'} Q_{\text{soft}}^{\theta}(\mathbf{s}_t, \mathbf{a}')\Big|_{\mathbf{a}'=\mathbf{a}_t}\right. \\ & \left. + \alpha \nabla_{\mathbf{a}'} \kappa\left(\mathbf{a}', f^{\phi}(\cdot; \mathbf{s}_t)\right)\Big|_{\mathbf{a}'=\mathbf{a}_t}\right]\end{aligned}$$

12 SAC-S算法

SAC-S算法[①]

```
初始化参数ψ, ψ̄, θ, ϕ
for 回合数 = 1, M do
    for 时步 = 1, t do
        根据a_t ∼ π_ϕ(a_t | s_t)采样动作a_t
        环境反馈奖励和下一个状态，s_{t+1} ∼ p(s_{t+1} | s_t, a_t)
        存储transition到经验回放中，D ← D ∪ {(s_t, a_t, r(s_t, a_t), s_{t+1})}
        更新环境状态s_{t+1} ← s_t
        更新策略：
        ψ ← ψ − λ_V ∇̂_ψ J_V(ψ)
        θ_i ← θ_i − λ_Q ∇̂_{θ_i} J_Q(θ_i) for i ∈ {1, 2}
        ϕ ← ϕ − λ_π ∇̂_ϕ J_π(ϕ)
        ψ̄ ← τψ + (1 − τ)ψ̄
    end for
end for
```

[①]Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

13 SAC算法

SAC算法[1]

```
初始化网络参数θ_1, θ_2以及φ
复制参数到目标网络θ̄_1 ← θ_1, θ̄_2 ← θ_2,
初始化经验回放D
for 回合数 = 1, M do
    重置环境，获得初始状态s_t
    for 时步 = 1, t do
        根据a_t ~ π_φ(a_t | s_t)采样动作a_t
        环境反馈奖励和下一个状态，s_{t+1} ~ p(s_{t+1} | s_t, a_t)
        存储transition到经验回放中，D ← D ∪ {(s_t, a_t, r(s_t, a_t), s_{t+1})}
        更新环境状态s_{t+1} ← s_t
        更新策略：
        更新Q函数，θ_i ← θ_i − λ_Q ∇̂_{θ_i} J_Q(θ_i) for i ∈ {1, 2}[2][3]
        更新策略权重，φ ← φ − λ_π ∇̂_φ J_π(φ) [4]
        调整temperature，α ← α − λ ∇̂_α J(α) [5]
        更新目标网络权重，θ̄_i ← τθ_i + (1 − τ)θ̄_i for i ∈ {1, 2}
    end for
end for
```

[2]Soft Actor-Critic Algorithms and Applications

[2] $J_Q(\theta)=\mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta\left(\mathbf{s}_t,\mathbf{a}_t\right)-\left(r\left(\mathbf{s}_t,\mathbf{a}_t\right)+\gamma\mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V_{\bar{\theta}}\left(\mathbf{s}_{t+1}\right)\right]\right)\right)^2\right]$

[3] $\hat{\nabla}_\theta J_Q(\theta)=\nabla_\theta Q_\theta\left(\mathbf{a}_t,\mathbf{s}_t\right)\left(Q_\theta\left(\mathbf{s}_t,\mathbf{a}_t\right)-\left(r\left(\mathbf{s}_t,\mathbf{a}_t\right)+\gamma\left(Q_{\bar{\theta}}\left(\mathbf{s}_{t+1},\mathbf{a}_{t+1}\right)-\alpha\log\left(\pi_\phi\left(\mathbf{a}_{t+1}\mid\mathbf{s}_{t+1}\right)\right)\right)\right)\right)$

[4] $\hat{\nabla}_\phi J_\pi(\phi)=\nabla_\phi\alpha\log\left(\pi_\phi\left(\mathbf{a}_t\mid\mathbf{s}_t\right)\right)+\left(\nabla_{\mathbf{a}_t}\alpha\log\left(\pi_\phi\left(\mathbf{a}_t\mid\mathbf{s}_t\right)\right)-\nabla_{\mathbf{a}_t}Q\left(\mathbf{s}_t,\mathbf{a}_t\right)\right)\nabla_\phi f_\phi\left(\epsilon_t;\mathbf{s}_t\right)$,$\mathbf{a}_t=f_\phi\left(\epsilon_t;\mathbf{s}_t\right)$

[5] $J(\alpha)=\mathbb{E}_{\mathbf{a}_t\sim\pi_t}\left[-\alpha\log\pi_t\left(\mathbf{a}_t\mid\mathbf{s}_t\right)-\alpha\overline{\mathcal{H}}\right]$

14 GAIL Algorithm

GAIL Algorithm[①]

1: Sample expert trajectories $\tau_E \sim \pi_E$, initial policy θ_0 and and discriminator D parametersω_0
2: **for** epoch $i = 1, 2, \cdots$ **do**
3: Sample policy trajectories$\tau_i \sim \pi_{\theta_i}$
4: Update the parameters ω_i of discriminator D using the gradient:

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_w \log\left(D_w(s,a)\right)\right] + \hat{\mathbb{E}}_{\tau_E}\left[\nabla_w \log\left(1 - D_w(s,a)\right)\right] \tag{1}$$

5: Update policy π_{θ_i} using the output of discriminator D as rewards for policy trajectories τ_i[②]
6: **end for**

[①]Generative Adversarial Imitation Learning

[②]The updating method of policy is closely related to the policy algorithm π_θ, for example PP0-Clip