

Deep Deterministic Policy Gradient DDPG

Background

- 同时学习 Q-Function 和 Policy π Algorithm
 - 使用 Off-Policy π Data 和 Bellman 方程 学习 Q-Function
 - 使用 Q-Function 学习 Policy 若已知最优的 Action-Value Function $Q^*(s, a)$, 则对于任何给定 state 都可以通过解决 $a^*(s) = \arg\max_a Q^*(s, a)$ 找到最优 action
- DDPG 是一种专门用于连续动作空间的 Q-Learning 计算 $\max_a Q^*(s, a)$ 的方式不同
 - 当 action 连续时, 无法穷举且计算 $\max_a Q^*(s, a)$ 非常耗费计算资源 每一个 action 都要运行一遍
 - 由于 action 连续, 可以认为 $Q^*(s, a)$ 对于 action a 微用 $\max_a Q(s, a) \approx Q(s, \mu(s)) \rightarrow \max_a Q(s, a)$

Key Equations

Q-Learning Side of DDPG

$$Q^*(s, a) = \mathbb{E}_{s' \sim P} [r(s, a) + \gamma \max_{a'} Q^*(s', a')] \rightarrow 描述 最优 Action-Value Function Q^*(s, a) 的 Bellman Equation$$

$s' \sim P$ 表示 next state s' 是 Environment 从 $P(s, a)$ 采样出的

- Bellman Equation 是学习 $Q^*(s, a)$ 的 Approximator 的起点, d 表示 s' 是否是 terminal state
- 对于 Neural Network $Q_\phi(s, a)$, 使用一系列 transitions (experience) (s, a, r, s', d) 训练参数中
- 损失函数使用 Mean-Squared Bellman Error (MSBE) Q_ϕ 距离满足 Bellman Equation 有多接近

$$L(\phi, D) = \mathbb{E}_{(s, a, r, s', d) \sim D} \left[\left(Q_\phi(s, a) - \left(r + \gamma (1-d) \max_{a'} Q_\phi(s', a') \right) \right)^2 \right] \rightarrow$$

当 d 为 True, s' 是 terminal state
 Q -Function 应表示 agent 在当前 state s 后不再有 reward

一个 D set 的先前经验

Trick One: Replay Buffer

- 所有训练 Deep 神经网络用于估计 $Q^*(s, a)$ 的 Algorithm 都使用 Experience Replay Buffer
 - Replay Buffer 必须足够大才能包含足够的 experience
 - 使用 recent data 会导致 overfit → experience 太多会导致学习速度下降
 - Replay Buffer 应该包含旧的 experience, 即使是过时的 policy 获得的
- Bellman Equation 不在意, 使用哪个 transition、如何选择 action、在给定 transition 后会是什么
- Optimal Q-Function 对于所有 transition 都满足 Bellman Equation
 - 在通过最小化 MSBE 获得 Q-Function Approximator 时, 经历过的任何 transitions 都是平等的

Trick Two: Target Networks

$$r + \gamma(1-d) \max_{a'} Q_\phi(s', a') \rightarrow \text{Target: } \text{最小化 MSBE loss 时, 希望令 Q-Function 更接近这个 target}$$

取决于希望训练得的参数中 → 导致训练不稳定

- 使用一组参数, 与中很接近, 但有一段时间的延迟 → 第二组 Target Network 的参数中 tang 滞后于中
 - DRN-based Algorithms 中, 每隔固定 steps 将 main Network 的参数复制到 target Network
 - DDPG 使用 Polyak Average, 每次更新 main Network 时都更新一次 target Network $\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1-\rho)\phi$,
 - Target Policy Network → 计算 Target 中的最大化 Action 计算最大化 $Q_{\phi_{\text{targ}}} \pi$ action
- 与 Target Q-Function 的计算方式相同, Polyak Averaging

$$L(\phi, D) = \mathbb{E}_{(s, a, r, s', d) \sim D} \left[\left(Q_\phi(s, a) - \left(r + \gamma(1-d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s')) \right) \right)^2 \right] \rightarrow$$

DDPG 和 Q-Learning 是通过最小化 MSBE loss 完成的
 \rightarrow MSBE 最小化通过 SGD (Stochastic Gradient Descent) 实现

Policy Learning Side of DDPG 希望学习到确定的 policy $\mu(s)$, 给出能最大化 $Q(s, a)$ 的 action
 → 由于 action space 连续且 Q-Function 可微, 可以使用 Gradient Descent 解决 $\max_{\theta} \mathbb{E}_{s \sim D} [Q_\phi(s, \mu_\theta(s))]$
 Q -Function 的参数视为常数 只与 policy 参数相关

Exploration vs. Exploitation

- 以 off-Policy π 方式训练, 一种确定性 policy
- 在训练时, 向 action 中加入 noise 使得 DDPG 更好地 explore
 - Uncorrelated Mean-zero Gaussian Noise
- 训练刚开始时固定步数, 采取 action 是对有效 action 的 Uniform Distribution 采样的

Soft Actor-Critic

Background 用于连续动作空间的 Off-Policy Algorithm

- 用一种 Off-Policy 的方式优化随机构性 Policy 结合随机构策优化和 DDPG 方法
- 同时结合了 Clipped Double-Q Trick 和 Target Policy Smoothing 得到 Policy 中的随机构性
- SAC的核心特征是 Entropy Regularization 训练 Policy 最大化 Expected Return 和 Entropy 间的平衡
→ 与 Exploration-Exploitation 的平衡密切相关 Entropy 增加导致更多的 Exploration
- → 不仅能够加速学习过程，还能防止 Policy 过早收敛至 Bad Local Optimum
- 可以通过改变 Policy 更新规则，将 SAC 用于离散动作空间

Key Equations

Entropy-Regularized Reinforcement Learning value Function 表达有所不同