Непрерывное пространство действий

Артем Латышев

01

Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG)

- DDPG одновременно обучает:
  - Функцию полезности $Q(s, a)$
  - Стратегию агента
- Использует off-policy данные и уравнение Беллмана для обновления функции полезности.
- Функция полезности используется для обучение стратегии.
- Подход тесно связан с Q-обучением, и мотивирован идей, если мы знаем оптимальную функцию полезности $Q^*(s, a)$, мы можем определить оптимальное действие:

$$a^*(s) = \arg\max_a Q^*(s, a).$$

Обучение Q-функции и стратегии

- DDPG чередует:
  - Обучение аппроксиматора для $Q^*(s, a)$ (Q-функции).
  - Обучение аппроксиматора для стратегии.
- Адаптирован специально для **непрерывных пространств действий**.
- Ключевая сложность: вычисление максимума по действиям в $\max_a Q^*(s, a)$.

Проблема максимизации в непрерывных пространствах

- В **дискретных пространствах действий**:
  - Максимизация проста: вычислить $Q(s, a)$ для всех возможных действий.
- В **непрерывных пространствах действий**:
  - Невозможно перебрать все действия.
  - Непосредственное решение $\max_a Q^*(s, a)$ вычислительно сложно.
- Предположение: $Q^*(s, a)$ **дифференцируема по действиям**.

Проблема максимизации в непрерывных пространствах

Поскольку пространство действий непрерывно, предполагается, что функция $Q^*(s,a)$ дифференцируема по аргументу действия. Это позволяет нам создать эффективное правило обучения стратегии $\mu(s)$ (также на основе градиентного спуска), используя данный факт.

Тогда вместо того, чтобы запускать трудоёмкую процедуру оптимизации каждый раз при вычислении $\max_a Q(s,a)$, мы можем использовать приближение:

$$\max_a Q(s,a) \approx Q(s,\mu(s))$$

Q-обучение в DDPG

- Уравнение Беллмана для оптимальной функции полезности действия $Q^*(s,a)$:

$$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

- Здесь $s' \sim P$ означает, что следующее состояние s' сэмплируется из распределения переходов среды $P(\cdot|s,a)$.
- Это уравнение является основой для обучения аппроксиматора $Q^*(s,a)$.

Обучение приближённой Q-функции

- Мы используем нейронную сеть $Q_\phi(s,a)$ с параметрами ϕ для аппроксимации $Q^*(s,a)$.
- Мы обучаем её, используя память прецедентов (replay buffer) $\mathcal{D}$, содержащий переходы (s,a,r,s',d).
- Функция потерь среднеквадратичной TD-ошибки имеет вид:

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

Объяснение условия терминального состояния

- Член $(1 - d)$ обеспечивает следующее:
  - Если $d = 1$ (терминальное состояние), то $\max_{a'} Q_\phi(s', a')$ игнорируется.
  - Если $d = 0$ (не терминальное состояние), то мы используем бутстрэппинг со следующим Q-значением.
- Работает в Python, где True $= 1$ и False $= 0$.
- Гарантирует, что Q-функция перестаёт накапливать награды после достижения терминального состояния.
- В новых версиях gym и gymnasium есть два флага — terminated и truncated.

Методики, используемые в Q-обучении

- Большинство алгоритмов Q-обучения, включая DQN и DDPG, минимизируют TD-ошибку.
- Используется две ключевых идеи:
    1. Целевые сети: медленно обновляемая сеть $Q_{\phi_{\text{targ}}}$ для стабилизации обучения.
    2. Память прецедентов: сохраняет предыдущие взаимодействия и производит их случайную выборку для уменьшения корреляции между обновлениями.

Обновления целевой (target) сети в DDPG

- В алгоритмах на основе DQN целевая сеть копируется из основной сети через фиксированное количество шагов.
- В DDPG целевая сеть обновляется при каждом обновлении основной сети с использованием усреднения Поляка (Polyak averaging):

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$

- Здесь ρ — гиперпараметр между 0 и 1 (обычно близкий к 1).
- Такой подход обеспечивает более плавное обновление целевой сети со временем.

Обработка максимума по действиям в DDPG

- В непрерывных пространствах действий вычисление $\max_a Q(s, a)$ является сложной задачей.
- Вместо прямой оптимизации DDPG использует **целевую сеть стратегии** для аппроксимации максимизирующего действия:

$$\tilde{a} = \mu_{\theta_{\text{targ}}}(s')$$

- Как и целевая Q-функция, целевая сеть стратегии обновляется через **усреднение Поляка**.
- Это гарантирует, что целевые действия меняются постепенно, повышая стабильность.

Q-обучение в DDPG

- Q-функция в DDPG обучается путем минимизации **среднеквадратичной TD-ошибки**:

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

- Здесь $\mu_{\theta_{\text{targ}}}$ — целевая сеть стратегии.
- Данная функция потерь минимизируется с помощью **стохастического градиентного спуска**.

Обучение стратегии в DDPG

- Цель — обучить **детерминированную стратегию** $\mu_\theta(s)$, которая максимизирует $Q_\phi(s, a)$.
- Поскольку $Q_\phi(s, a)$ дифференцируема по a, мы можем **напрямую оптимизировать стратегию**:

$$\max_\theta \mathop{\mathrm{E}}_{s \sim \mathcal{D}} \left[Q_\phi(s, \mu_\theta(s)) \right]$$

- Используем **градиентный подъём** для обновления параметров стратегии θ.
- Параметры Q-функции ϕ рассматриваются как **константы** во время оптимизации стратегии.

Исследование против использования в DDPG

- DDPG обучает **детерминированную стратегию**.
- Полностью детерминированная стратегия не обеспечивает достаточного исследования на начальных этапах.
- Для стимулирования исследования мы добавляем **шум** к действиям во время обучения.
- В оригинальной статье DDPG предлагался **шум Орнштейна-Уленбека (OU)**, но современные исследования показывают, что:
  - **Некоррелированный гауссовский шум** (с нулевым средним) работает не хуже.
  - Гауссовский шум проще и чаще используется на практике.
- Опционально, масштаб шума можно **уменьшать со временем** для улучшения эффективности стратегии.
- В нашей реализации **шум остаётся фиксированным** на протяжении всего обучения.

DDPG Pseudocode

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
    Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
    Execute a in the environment
    Observe next state s', reward r, and done signal d to indicate whether s' is terminal
    Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
    If s' is terminal, reset environment state.
    if it's time to update **then**
        for however many updates **do**
            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
            Compute targets

$$y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

$$y(r, s', d) = r + \gamma(1-d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

Update Q-function by one step of gradient descent using

$$\nabla_{\phi} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi}(s,a) - y(r,s',d) \right)^2$$

Update policy by one step of gradient ascent using

$$\nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} Q_{\phi}(s, \mu_{\theta}(s))$$

Update target networks with

$$\begin{aligned} \phi_{\text{targ}} &\leftarrow \rho \phi_{\text{targ}} + (1-\rho)\phi \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1-\rho)\theta \end{aligned}$$

end for

end if

until convergence

02

Twin Delayed DDPG

Ограничения DDPG

- Несмотря на то, что DDPG может демонстрировать высокую производительность, алгоритм **часто оказывается неустойчивым** к изменению гиперпараметров.
- Распространённая проблема — **завышение оценок Q-функции**, приводящее к нестабильности стратегии.
- Обучаемая стратегия использует ошибки в Q-функции, что ухудшает производительность.
- **Twin Delayed DDPG (TD3)** решает эти проблемы с помощью трёх ключевых методик.

Методика первая: Clipped Double-Q Learning

- TD3 обучает **две Q-функции** вместо одной (отсюда "twin" - двойной).
- Использует **минимум из двух Q-значений** для формирования целевых значений в функции потерь TD-ошибки.
- Это уменьшает **смещение завышения оценок** в Q-обучении.

Методика вторая: Delayed Policy Updates

- TD3 обновляет **стратегию и целевые сети реже**, чем Q-функцию.
- В статье рекомендуется **одно обновление стратегии на каждые два обновления Q-функции**.
- Это снижает **вариативность обновлений стратегии**, делая обучение более стабильным.

Методика третья: Target Policy Smoothing

- **Шум добавляется** к целевому действию, чтобы предотвратить переобучение на ошибках Q-функции.
- Это **сглаживает Q-значения** при небольших изменениях действий, повышая устойчивость.
- Помогает избежать **использования некорректных резких пиков Q-функции**.

Ключевые уравнения: Target Policy Smoothing

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

- Целевые действия **зашумляются ограниченным гауссовским шумом**.
- Затем шумное действие **ограничивается допустимым диапазоном действий**.
- Такой подход предотвращает негативное влияние **чрезмерно резких градиентов Q-функции** на обучение.

Clipped Double-Q Learning

→ Обе Q-функции используют **единую цель**, определяемую **меньшим Q-значением**:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')).$$

$$L(\phi_1, \mathcal{D}) = \mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}} \left[\left(Q_{\phi_1}(s, a) - y(r, s', d) \right)^2 \right]$$

$$L(\phi_2, \mathcal{D}) = \mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}} \left[\left(Q_{\phi_2}(s, a) - y(r, s', d) \right)^2 \right]$$

Обучение стратегии в TD3

- Стратегия обучается путём **максимизации Q-функции**:

$$\max_{\theta} \mathop{\mathrm{E}}_{s \sim \mathcal{D}} \left[Q_{\phi_1}(s, \mu_\theta(s)) \right].$$

- **Так же, как в DDPG**, но с важным отличием:
- В TD3 **стратегия обновляется реже**, чем Q-функции.
- Это **снижает нестабильность обучения**, вызванную быстрыми изменениями целей.

Algorithm 3 Twin Delayed DDPG

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$

repeat

  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$

  Execute a in the environment

  Observe next state s', reward r, and done signal d to indicate whether s' is terminal

  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

  If s' is terminal, reset environment state.

  if it's time to update **then**

    for j in range(however many updates) **do**

      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

      Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d) \right)^2 \qquad \text{for } i = 1, 2$$

if $j \mod \texttt{policy\_delay} = 0$ **then**

Update policy by one step of gradient ascent using

$$\nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_{\theta}(s))$$

Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$$

end if

end for

end if

until convergence

03

Soft Actor-Critic

Soft Actor-Critic

Алгоритм Soft Actor-Critic (SAC) заполняет разрыв между стохастической оптимизацией градиента стратегии и подходами в стиле DDPG. Он не является прямым преемником TD3 (поскольку был опубликован примерно в то же время), но включает в себя метод clipped double-Q. Кроме того, благодаря внутренней стохастичности стратегии в SAC, он также выигрывает от сглаживания целевой стратегии.

Soft Actor-Critic

Ключевой особенностью SAC является энтропийная регуляризация.

Стратегия обучается для максимизации баланса между ожидаемым вознаграждением и энтропией — мерой случайности в стратегии. Это тесно связано с балансом между исследованием и использованием: увеличение энтропии приводит к большему исследованию среды, что впоследствии может ускорить обучение. Это также может предотвратить преждевременную сходимость стратегии к локальному оптимуму.

Энтропийно-регуляризованное обучение с подкреплением

Энтропия — это величина, которая, грубо говоря, показывает, насколько случайна случайная величина. Если монета смещена так, что почти всегда выпадает орёл, она имеет низкую энтропию; если она сбалансирована и имеет равные шансы для любого исхода, она имеет высокую энтропию.

Пусть x — случайная величина с функцией вероятности или плотности P. Энтропия H величины x вычисляется по её распределению P следующим образом:

$$H(P) = \mathbb{E}_{x \sim P} \left[-\log P(x) \right].$$

Энтропийно-регуляризованное обучение с подкреплением

Агент получает дополнительную награду на каждом шаге, пропорциональную энтропии стратегии в этот момент времени. Это преобразует задачу RL в:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t, a_t, s_{t+1}) + \alpha H\big(\pi(\cdot|s_t)\big)\right)\right],$$

где $\alpha > 0$ — коэффициент балансирующией исследование.

Предполагаем бесконечный горизонт с дисконтированием.

Энтропийно-регуляризованное обучение с подкреплением

Функция полезности в этом подходе модифицируется для включения энтропийных бонусов на каждом шаге:

$$V^{\pi}(s) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}\Big(R(s_t,a_t,s_{t+1}) + \alpha H\big(\pi(\cdot|s_t)\big)\Big)\middle| s_0 = s\right].$$

Функция Q^{π} изменена для включения энтропийных бонусов на всех шагах, **кроме первого**:

$$Q^{\pi}(s,a) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^{t}H\big(\pi(\cdot|s_t)\big)\middle| s_0 = s, a_0 = a\right].$$

Энтропийно-регуляризованная Q-функция

При таких определениях V^π и Q^π связаны соотношением:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi}\left[Q^\pi(s,a)\right] + \alpha H\big(\pi(\cdot|s)\big).$$

Уравнение Беллмана для Q^π имеет вид:

$$Q^\pi(s,a) = \mathbb{E}_{s' \sim P}\left[R(s,a,s') + \gamma \mathbb{E}_{a' \sim \pi}\left[Q^\pi(s',a') + \alpha H\big(\pi(\cdot|s')\big)\right]\right].$$

Что упрощается до:

$$Q^\pi(s,a) = \mathbb{E}_{s' \sim P}\left[R(s,a,s') + \gamma V^\pi(s')\right].$$

Soft Actor-Critic и TD3

SAC одновременно обучает стратегию π_θ и две Q-функции Q_{ϕ_1}, Q_{ϕ_2}. Существуют две стандартных версии SAC: одна использует фиксированный коэффициент энтропийной регуляризации α, а другая поддерживает энтропийное ограничение, варьируя α в процессе обучения.

Q-функции обучаются аналогично TD3, но с несколькими ключевыми отличиями.

Сначала о сходствах:

- Как и в TD3, обе Q-функции обучаются минимизацией MSBE, регрессируя к единой общей цели.
- Как и в TD3, общая цель вычисляется с использованием целевых Q-сетей, которые получаются полиаковским усреднением параметров Q-сетей в процессе обучения.
- Как и в TD3, общая цель использует метод clipped double-Q.

Soft Actor-Critic и TD3

В чём различия?

- В отличие от TD3, цель также включает член, связанный с использованием энтропийной регуляризации в SAC.
- В отличие от TD3, действия для следующего состояния, используемые в цели, берутся из текущей стратегии, а не из целевой стратегии.
- В отличие от TD3, в SAC нет явного сглаживания целевой стратегии. TD3 обучает детерминированную стратегию и достигает сглаживания путём добавления случайного шума к действиям для следующего состояния. SAC обучает стохастическую стратегию, и шум от этой стохастичности достаточен для достижения схожего эффекта.

Аппроксимация матожидания

Прежде чем мы приведем окончательный вид Q-потерь, давайте обсудим, как появляется вклад от энтропийной регуляризации. Мы начнём с рекурсивногои уравнения Беллмана для энтропийно-регуляризованного Q^π, приведённого ранее, и слегка перепишем его, используя определение энтропии:

$$
\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}_{s' \sim P, a' \sim \pi}\left[R(s,a,s') + \gamma\left(Q^\pi(s',a') + \alpha H\big(\pi(\cdot|s')\big)\right)\right] \\
&= \mathbb{E}_{s' \sim P, a' \sim \pi}\left[R(s,a,s') + \gamma\left(Q^\pi(s',a') - \alpha \log \pi(a'|s')\right)\right].
\end{aligned}
$$

Аппроксимация матожидания

Правая часть уравнения Беллмана представляет собой математическое ожидание по следующим состояниям (которые берутся из памяти прецедентов) и следующим действиям (которые берутся из текущей стратегии, а не из памяти).

Поскольку это математическое ожидание, мы можем аппроксимировать его с помощью выборок:

$$Q^{\pi}(s,a) \approx r + \gamma \left(Q^{\pi}(s', \tilde{a}') - \alpha \log \pi(\tilde{a}'|s') \right), \quad \tilde{a}' \sim \pi(\cdot|s').$$

Эта выборочная аппроксимация используется на практике для эффективной оценки Q-значений.

Q-потери в SAC

SAC устанавливает в качестве функции потерь среднеквадратичную TD-ошибку для каждой Q-функции.

Единственное, что остаётся неопределённым — это то, какая Q-функция используется для вычисления выборочного обновления. Как и в TD3, SAC использует метод clipped double-Q и берёт минимальное Q-значение из двух аппроксиматоров Q.

Q-потери в SAC

Объединяя всё вместе, функции потерь для Q-сетей в SAC выглядят так:

$$L(\phi_i, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathbb{E}}\left[\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2\right],$$

где цель задаётся как:

$$y(r,s',d) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}(s',\tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right),$$

где $\tilde{a}' \sim \pi_\theta(\cdot|s')$.

Обучение стратегии

Стратегия должна в каждом состоянии действовать так, чтобы максимизировать ожидаемую отдачу плюс ожидаемую будущую энтропию.

То есть она должна максимизировать $V^\pi(s)$, что раскрывается как:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{a\sim\pi}\left[Q^\pi(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right) \\ &= \mathbb{E}_{a\sim\pi}\left[Q^\pi(s,a) - \alpha\log\pi(a|s)\right]. \end{aligned}$$

Трюк с репараметризацией

Способ оптимизации стратегии использует **трюк с репараметризацией**, при котором выборка из $\pi_\theta(\cdot|s)$ извлекается путём вычисления детерминированной функции от состояния, параметров стратегии и независимого шума.

Для иллюстрации: следуя статье SAC, мы используем сжатую гауссову стратегию, что означает, что выборки получаются согласно:

$$\tilde{a}_\theta(s, \xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s) \odot \xi\right), \quad \xi \sim \mathcal{N}(0, I).$$

Этот подход обеспечивает плавное исследование, сохраняя при этом ограниченные выходы действий.

Ключевые отличия стратегии SAC

Эта стратегия имеет два ключевых отличия от стратегий, используемых в других алгоритмах оптимизации:

1. **Функция сжатия:** $\tanh$ в стратегии SAC гарантирует, что действия ограничены конечным диапазоном. Это отсутствует в стратегиях VPG, TRPO и PPO. Также изменяет распределение: до $\tanh$ стратегия SAC является факторизованной гауссовой, как и другие, но после $\tanh$ — нет.

2. **Зависящие от состояния стандартные отклонения:** В VPG, TRPO и PPO логарифмы стандартных отклонений представлены векторами параметров, не зависящими от состояния. В SAC логарифмы стандартных отклонений являются выходами нейронной сети, что делает их зависящими от состояния.

Трюк с репараметризацией и потери стратегии

Трюк с репараметризацией позволяет нам переписать математическое ожидание по действиям в математическое ожидание по шуму:

$$\mathbb{E}_{a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a)-\alpha\log\pi_\theta(a|s)\right]=\mathbb{E}_{\xi\sim\mathcal{N}}\left[Q^{\pi_\theta}(s,\tilde{a}_\theta(s,\xi))-\alpha\log\pi_\theta(\tilde{a}_\theta(s,\xi)|s)\right].$$

Трюк с репараметризацией и потери стратегии

Чтобы получить потери стратегии, мы заменяем Q^{π_θ} на функциональный аппроксиматор. В отличие от TD3, который использует только Q_{ϕ_1}, SAC использует минимум из двух Q-аппроксиматоров:

$$\max_\theta \mathbb{E}_{s\sim\mathcal{D},\xi\sim\mathcal{N}}\left[\min_{j=1,2} Q_{\phi_j}(s,\tilde{a}_\theta(s,\xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi)|s)\right].$$

Это похоже на DDPG и TD3, за исключением метода min-double-Q, стохастичности и энтропийного члена.

SAC Псевдокод

Algorithm 5 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
    Observe state s and select action $a \sim \pi_\theta(\cdot|s)$ and execute a in the environment
    Observe next state s', reward r, and done signal d to indicate whether s' is terminal
    Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
    If s' is terminal, reset environment state.
    if it's time to update **then**
        for j in range(however many updates) **do**
            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
            Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \qquad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

$$y(r, s', d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \qquad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2 \qquad \text{for } i = 1,2$$

Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \Big(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s) |\, s\right)\Big),$$

where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.

Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1,2$$

end for

end if

until convergence

Вопросы?