ML Advanced

Actor-Critic

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в учебной группе

Задаем вопрос
в чат

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

Тема вебинара

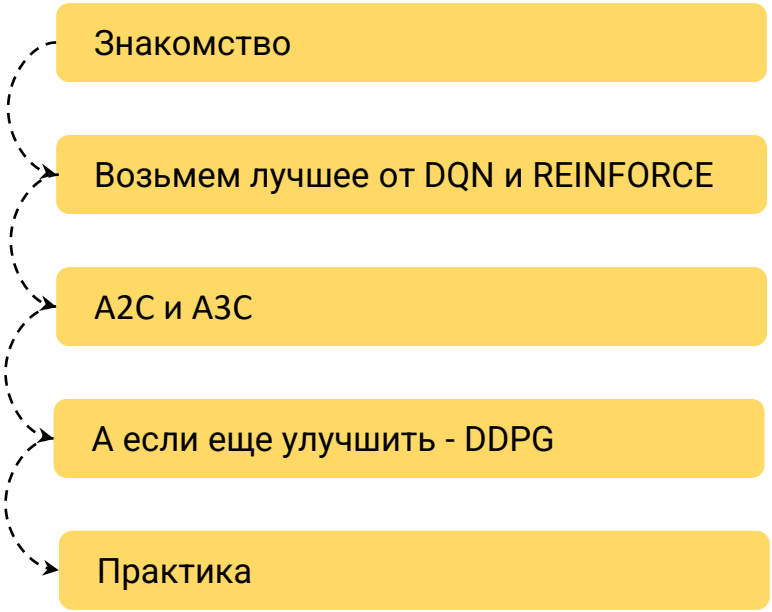
ML Advanced Actor-Critic

Андрей Канашов

Team Lead Data Scientist в ПИК

- Ценообразование и тарификация
- Рекомендательные системы
- Прогнозирование ключевых метрик
- Анализ клиентского поведения

Маршрут вебинара



Цели вебинара

К концу занятия вы сможете

1. Собрать лучшие части от алгоритмов DQN и REINFORCE
2. Понять алгоритмы A2C, A3C и DDPG
3. Понять решение задач в непрерывных средах

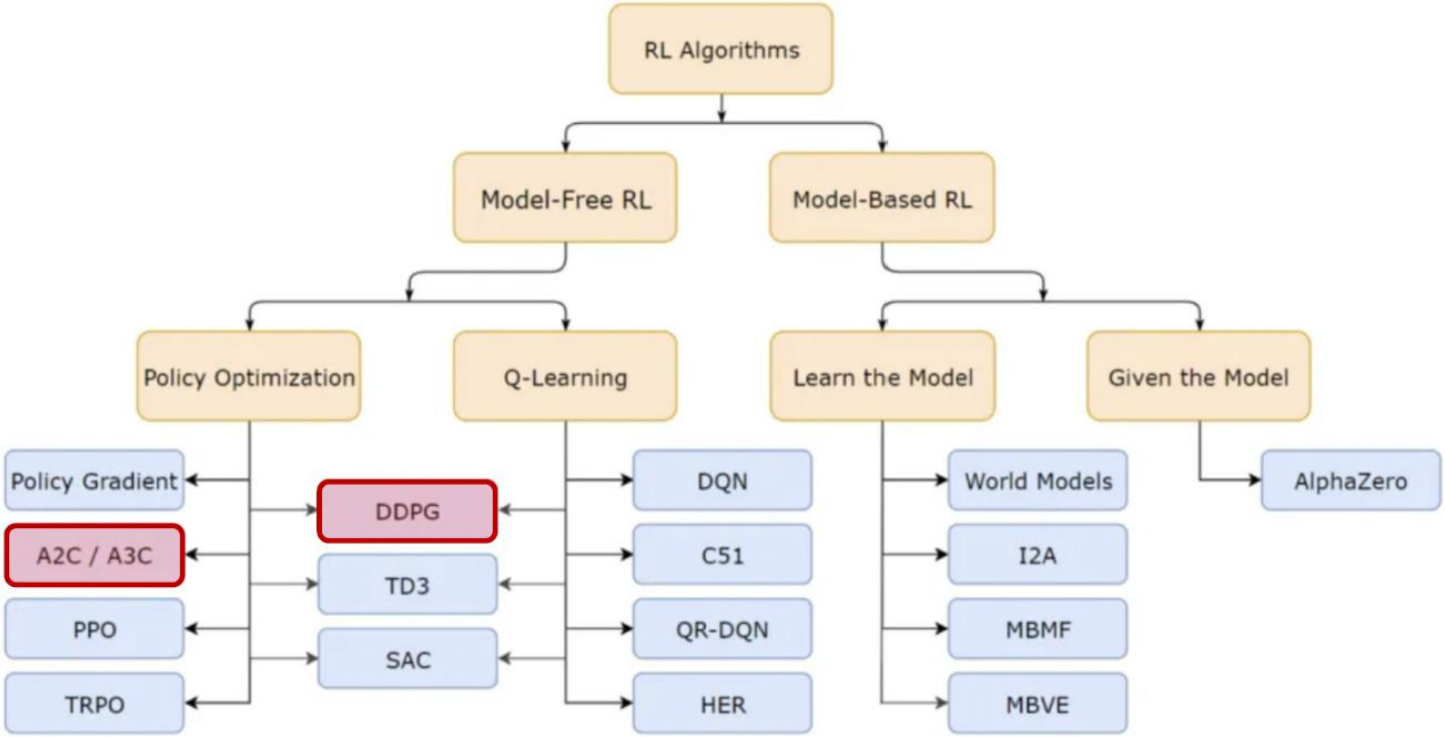
Смысл

Зачем вам это уметь

1. Уметь решать задачи в средах с непрерывными действиями и состояниями
2. Управлять вариативностью и робастностью обучения
3. Иметь базу для перехода к задачам мультиагентного обучения
4. Иметь базу для перехода к задачам частично-наблюдаемых процессов

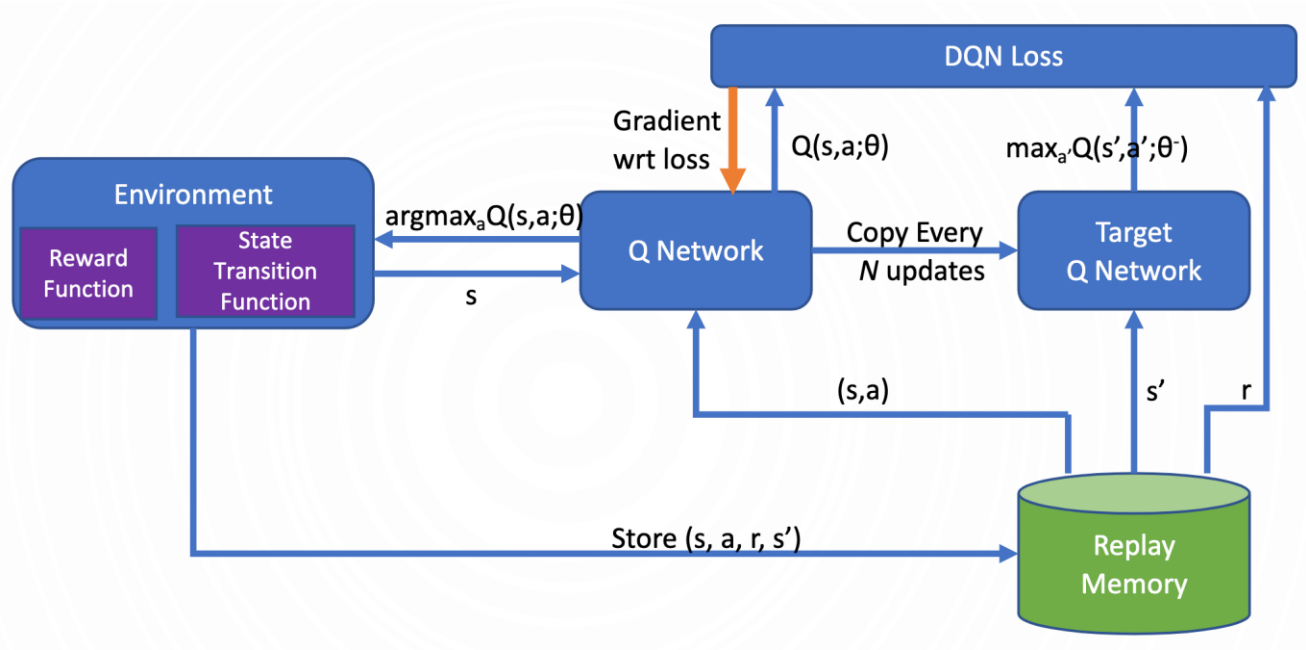
Семейство алгоритмов

Algorithms


RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

Алгоритм Actor-Critic

DQN


DQN Loss
Gradient
wrt loss
Q(s,a;θ)
max$_{a'}$Q(s',a';θ$^-$)
Environment
Reward Function
State Transition Function
argmax$_a$Q(s,a;θ)
s
Q Network
Copy Every
N updates
Target Q Network
(s,a)
s'
r
Store (s, a, r, s')
Replay Memory

DQN

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)\right)$$

Loss:

$$Loss(\theta) = \left(R_t + \gamma \max_{a'} Q^{\theta}(S_{t+1}, a') - Q^{\theta}(S_t, A_t)\right)^2$$

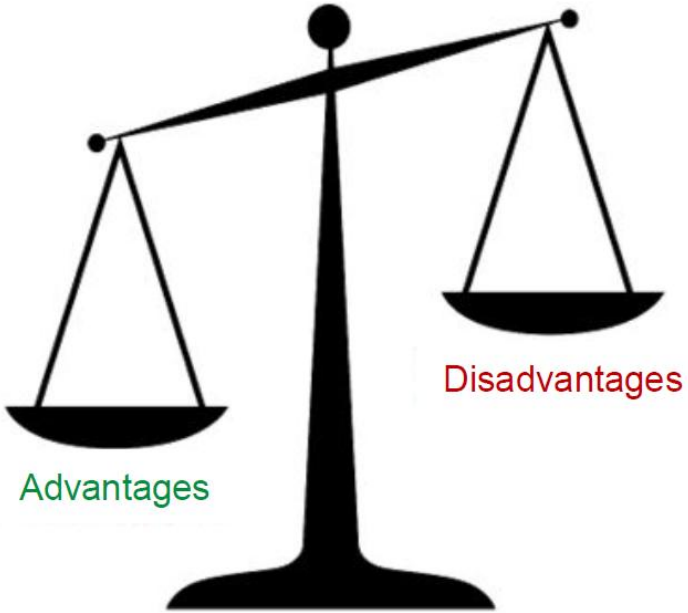
$$\nabla_{\theta} Loss(\theta) \approx -2\left(R_t + \gamma \max_{a'} Q^{\theta}(S_{t+1}, a') - Q^{\theta}(S_t, A_t)\right)\nabla_{\theta} Q^{\theta}(S_t, A_t)$$

$$\theta \leftarrow \theta - \alpha\nabla_{\theta} Loss(\theta)$$

DQN – достоинства и недостатки

Off-policy

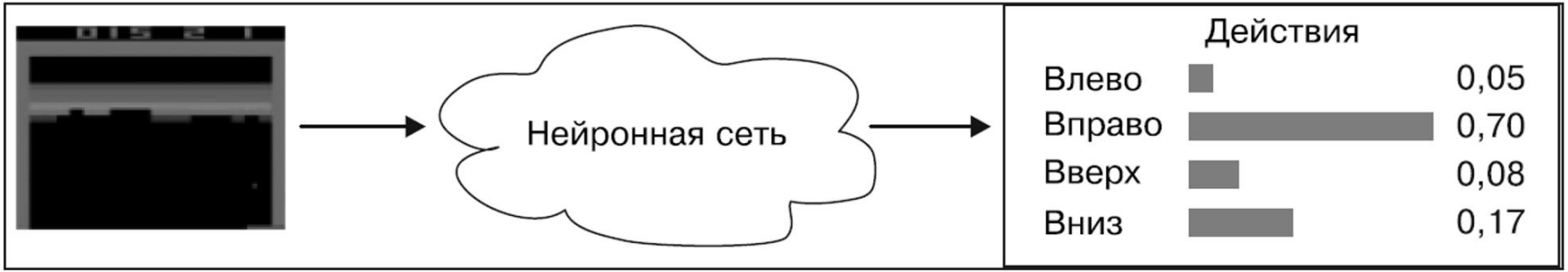
Использование
Experience replay

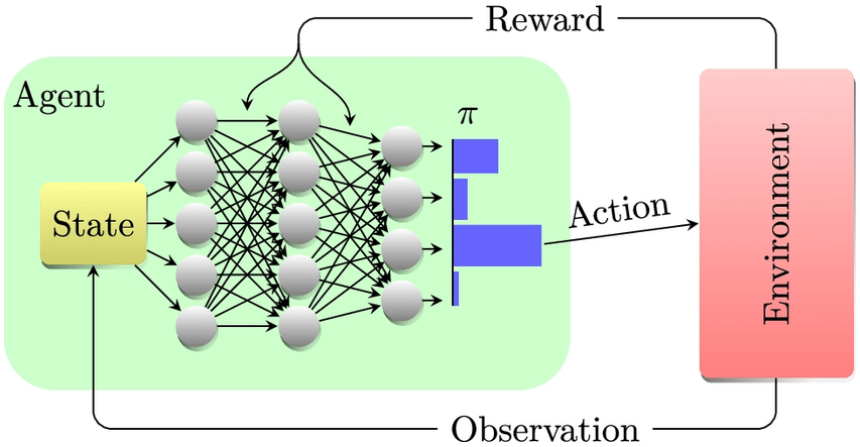


Дискретное
пространство
действий

Сложно обучать
Q-функцию

REINFORCE - алгоритм





REINFORCE - алгоритм

- Задаем начальное приближение политики $\pi^{\eta}(a|s)$
- Запускаем обучение. Для каждого эпизода:
  - Действуя по текущей политике π^{η} получаем траекторию $\tau = (S_0, S_1, \dots, S_T)$, награды $r = (R_0, R_1, \dots, R_{T-1})$ и определяем $g = (G_0, G_1, \dots, G_{T-1})$:

$$\mathrm{G}_t = \sum_{i=t}^{T} \gamma^{i-t} R_i$$

  - Следуя по траектории обновляем веса модели по правилу:

$$\eta \leftarrow \eta - \alpha \nabla_{\eta} \ln \pi^{\eta}(A_t|S_t) G_t$$

REINFORCE – достоинства и недостатки

Непрерывное пространство действий

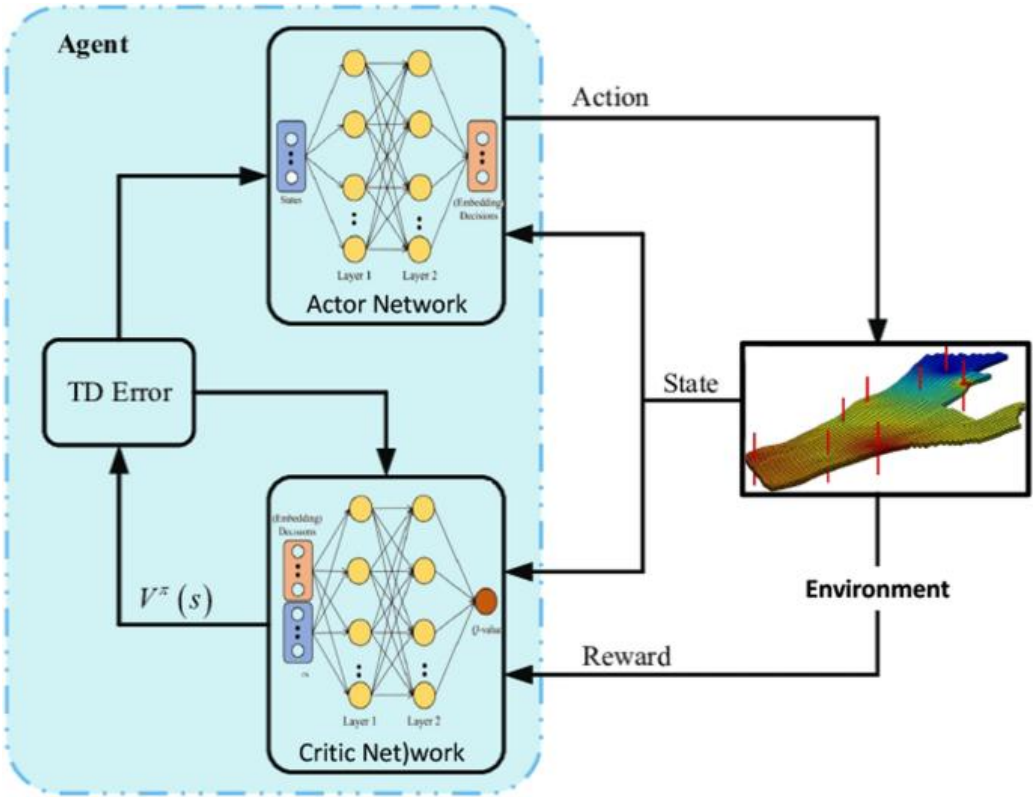
Обучаем сразу политику

On-policy

Не можем использовать Experience replay

Не устойчив

Actor-Critic



Actor-Critic

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(A_t, S_t) G_t \right] = \mathbb{E}_\tau \left[\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(A_t, S_t) \right] \mathbb{E}_\tau[G] =$$

$$\mathbb{E}_\tau \left[\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(A_t, S_t)\, Q(S_t, A_t) \right]$$

- Политику приближаем по алгоритму REINFORCE
- Q-функцию обновляем по DQN

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(A_t, S_t)\, Q(S_t, A_t) \right]$$

actor (REINFORCE)

critic (DQN)

Actor-Critic алгоритм

1. Инициализируем случайным образом сети actor $\pi^{\mu}(a|s)|\theta^{\mu}$ и critic $Q^{\theta}(s,a|\theta^{Q})$ весами Q^{θ} и θ^{μ} и целевые сети Q' и π': $\theta^{Q'} \leftarrow \theta^{Q}, \theta^{\mu'} \leftarrow \theta^{\mu}$
2. Устанавливаем число эпизодов обучения *M* и для каждого эпизода выполняем:
   - Действуем текущей политикой и получаем состояние s_1
   - Находясь в состоянии s_t в соответствии с текущей политикой выбираем действие $a_t = \pi^{\mu}(s_t|\theta^{\mu})$
   - Выполняем действие a_t, получаем награду r_t и переходим в следующее состояние s_{t+1}
   - Имея s_{t+1} в соответствии с политикой выбираем действие a_{t+1}
   - Вычисляем Loss для обновления весов:

$$Loss(\theta^{Q}) = \left(r_t + \gamma Q^{\theta}(s_{t+1}, a_{t+1}) - Q^{\theta}(s_t, a_t)\right)^2$$
$$Loss(\theta^{\mu}) = \ln \pi^{\mu}(a_t|s_t) Q^{\theta}(s_t, a_t)$$

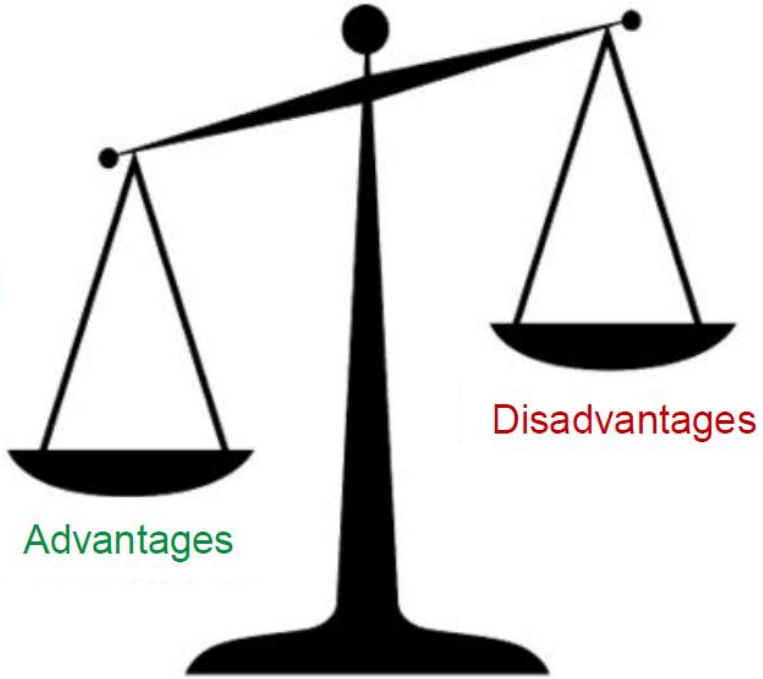
   - Обновляем веса: $\theta^{Q} \leftarrow \theta^{Q} - \alpha \nabla_{\theta^{Q}} Loss(\theta^{Q}), \theta^{\mu} \leftarrow \theta^{\mu} + \beta \nabla_{\theta^{\mu}} Loss(\theta^{\mu})$
   - Обновляем целевые сети

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

Actor-Critic – достоинства и недостатки

Непрерывное пространство действий

Стабильнее обучение за счет использования двух ветвей $\pi(a|s)$ и $Q(s,a)$



On-policy

Не можем использовать Experience replay

Сложно обучать Q-функцию

Алгоритм A2C

Advantage Actor-Critic (A2C)

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(A_t, S_t)\, A(S_t, A_t) \right]$$

Введем функцию преимущества (advantage): $A(S_t, A_t) = Q(S_t, A_t) - V(S_t)$

$$Q(S_t, A_t) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})], \quad \rightarrow \quad A(S_t, A_t) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(A_t, S_t) (R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \right]$$

Actor

Advantage Critic

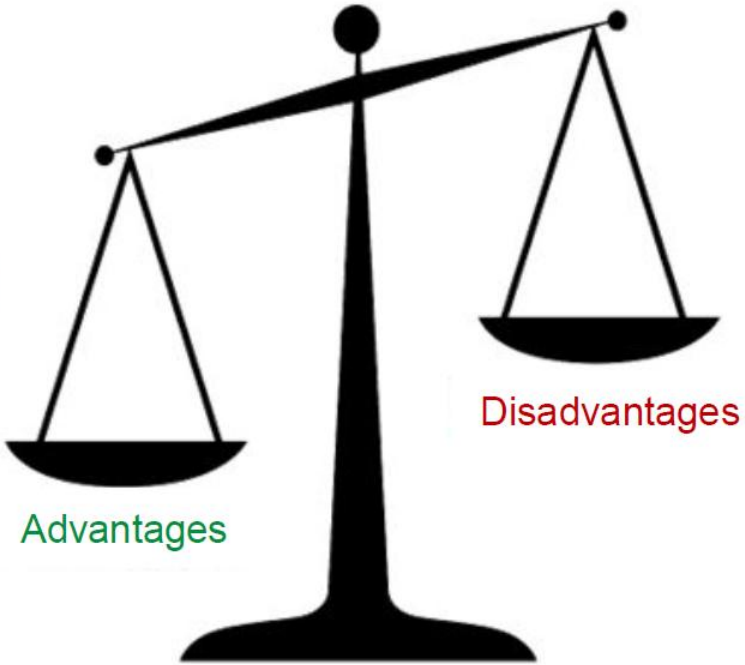
Advantage Actor-Critic (A2C) алгоритм

1. Инициализируем случайным образом сети actor $\mu(s|\theta^{\mu})$ и critic $V(s\ |\theta^{V})$ весами V^{θ} и θ^{μ} и целевые сети V' и μ': $\theta^{V'} \leftarrow \theta^{V}, \theta^{\mu'} \leftarrow \theta^{\mu}$
2. Устанавливаем число эпизодов обучения *M* и для каждого эпизода выполняем:
   - Действуем текущей политикой и получаем состояние s_1
   - Проходим по всем возможным действиям от 1 до *T*:
     - Выбираем действие $a_t = \mu(s_t|\theta^{\mu})$ в соответствии с текущей политикой
     - Выполняем действие a_t, получаем награду r_t и переходим в следующее состояние s_{t+1}
     - Обновляем политику используя текущее приближение для V
     $$\mu \leftarrow \mu + \alpha V(s|\theta^{V})\nabla_{\mu}\ln \pi_{\mu}(a|s)$$
     - Вычисляем таргеты $y_i = r_i + \gamma V'(S_{t+1}|\theta^{V'})\ - V(S_t|\theta^{V'})$
     - Обновляем V функцию на базе полученных таргетов $L = \left(y_i - V(s_i|\theta^{V})\right)^2, \theta^{V} \leftarrow \theta^{V}\ - \alpha\nabla_{\theta^{V}}$
     - Обновляем целевые сети
     $$\theta^{V'} \leftarrow \tau\theta^{V} + (1-\tau)\theta^{V'}$$
     $$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

A2C – достоинства и недостатки

Непрерывное пространство действий

Стабильнее обучение за счет использования двух ветвей $\pi(a|s)$ и $V(s)$

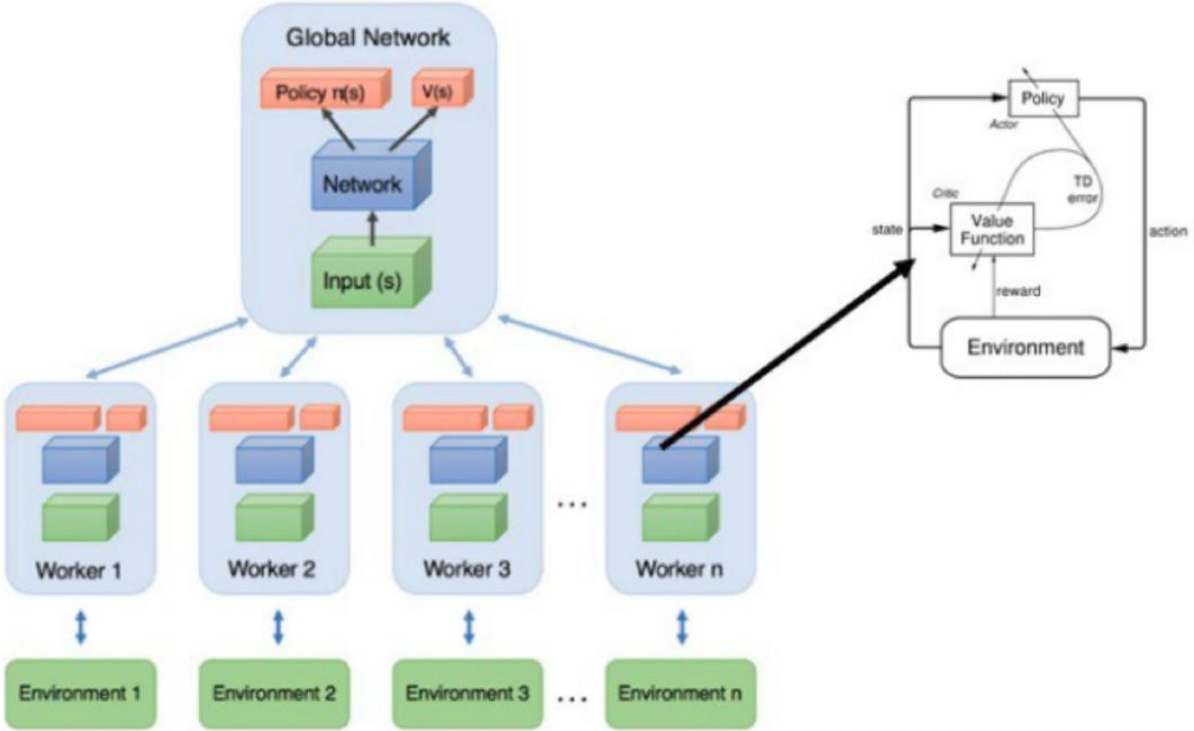


On-policy

Не можем использовать Experience replay

Алгоритм АЗС

Asynchronous Advantage Actor-Critic (A3C)

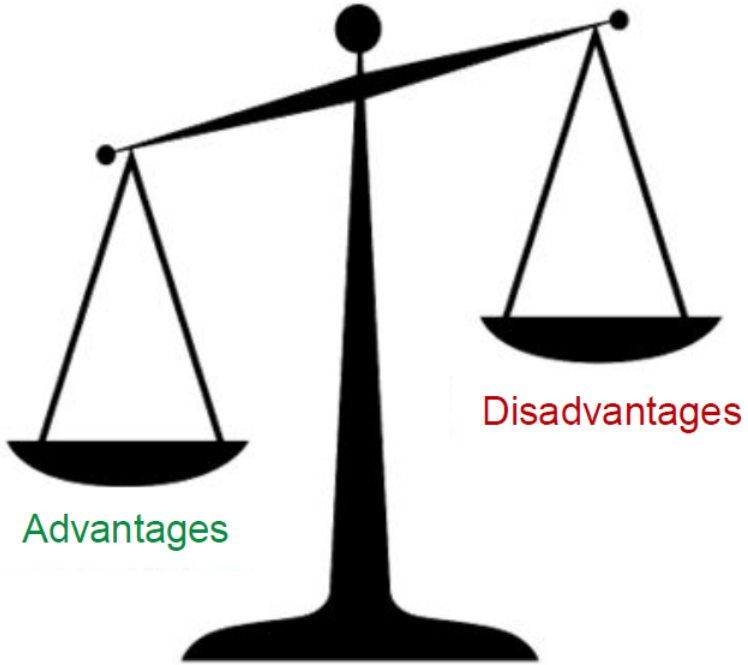


A3C – достоинства и недостатки

Непрерывное пространство действий

Off-policy

Стабильнее обучение за счет использования двух ветвей $\pi(a|s)$ и $V(s)$

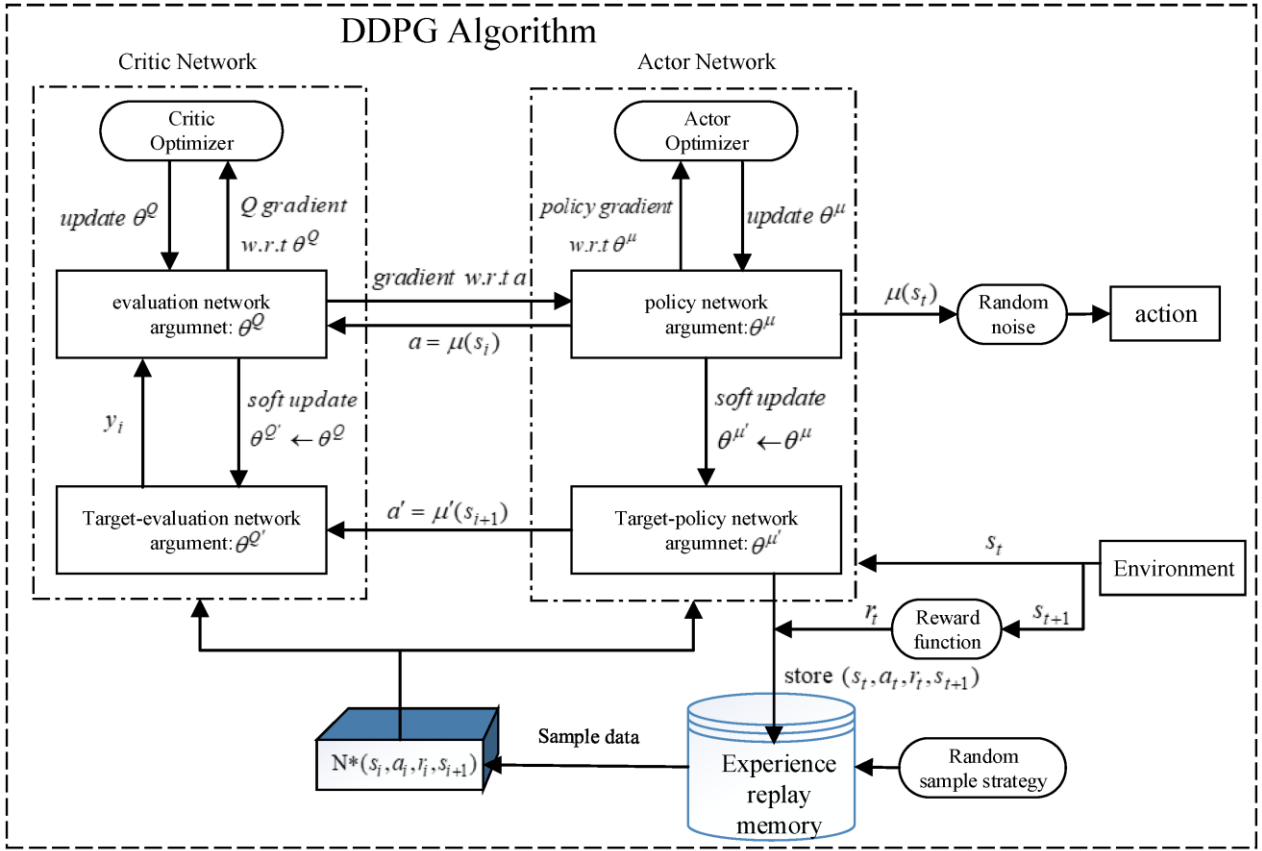


Многопроцессорная обработка

Нельзя использовать Experience replay

Алгоритм DDPG

DDPG


DDPG Algorithm
Critic Network
Actor Network
Critic Optimizer
Actor Optimizer
update θ^Q
Q gradient w.r.t θ^Q
policy gradient w.r.t θ^μ
update θ^μ
gradient w.r.t a
$a = \mu(s_i)$
evaluation network argumnet: θ^Q
policy network argument: θ^μ
$\mu(s_t)$
Random noise
action
y_i
soft update $\theta^{Q'} \leftarrow \theta^Q$
soft update $\theta^{\mu'} \leftarrow \theta^\mu$
$a' = \mu'(s_{i+1})$
Target-evaluation network argument: $\theta^{Q'}$
Target-policy network argumnet: $\theta^{\mu'}$
s_t
Environment
r_t
Reward function
s_{t+1}
store (s_t, a_t, r_t, s_{t+1})
N*(s_i, a_i, r_i, s_{i+1})
Sample data
Experience replay memory
Random sample strategy

DDPG – Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) - это алгоритм, который одновременно обучает Q-функцию и политику. Он использует данные вне политики и уравнение Беллмана для изучения Q-функции и использует Q-функцию для изучения политики.
“Continuous Control With Deep Reinforcement Learning” (Lillicrap et al, 2015)

Q-функция оценивает ожидаемое суммарное вознаграждение за выполнение определенного действия в данном состоянии, а сеть политики производит действия, максимизирующие Q-значение.

1. Experience replay buffer
2. Actor & Critic network updates
3. Target network updates
4. Exploration

https://arxiv.org/pdf/1509.02971

Deterministic Policy Gradient Theorem

Policy Gradient Theorem:

$$\nabla_\eta J(\eta) = \mathbb{E}_{\substack{s\sim\rho_{\pi^\eta} \\ a\sim\pi^\eta}}\left[\nabla_\eta \ln \pi^\eta(a|s) q_{\pi^\eta}(s,a)\right] \approx \nabla_\eta \ln \pi^\eta(a|s) q_{\pi^\eta}(s,a)$$

Будем искать оптимальную детерминированную политику $\pi^\eta(s) \approx \pi^*(s)$:

$$\nabla_\eta J(\eta) = \mathbb{E}_{s\sim\rho_{\pi^\eta}}\left[\nabla_\eta q_\pi(s,\pi^\eta(s)\right] \approx \nabla_\eta \left(\frac{1}{N}\sum_{i=1}^{N} Q^\theta(s_i,\pi^\eta(s_i))\right)$$

где N – размер батча.

Для аппроксимации функции Q используем уравнение Беллмана.

DDPG – Deep Deterministic Policy Gradient

1. Инициализируем случайным образом сети actor $\mu(s|\theta^\mu)$ и critic $Q(s,a|\theta^Q)$ весами Q^θ и θ^μ и целевые сети Q' и μ': $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
2. Инициализируем *Replay Buffer - R*
3. Устанавливаем число эпизодов обучения *M* и для каждого эпизода выполняем:
   - Инициализируем случайный процесс (шум) для исследования пространства действий $\mathcal{N}$ (Орнштейна-Уленбека)
   - Действуем текущей политикой и получаем состояние s_1
   - Проходим по всем возможным действиям от 1 до *T*:
     - Выбираем действие $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ в соответствии с текущей политикой и шумом (разведка)
     - Выполняем действие a_t, получаем награду r_t и переходим в следующее состояние s_{t+1}
     - Помещаем в память $(s_t, a_t, r_t, s_{t+1}) \rightarrow R$
     - Получаем из памяти случайный минибатч $(s_i, a_i, r_i, s_{i+1})_{i\in[1,K]}$
     - Получаем таргеты $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
     - Обновляем критика используя лосс: $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$, $\theta^Q \leftarrow \theta^Q - \alpha\nabla_{\theta^Q} L$
     - Обновляем актора используя policy gradient:
       $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{\substack{s=s_i \\ a=\mu(s_i)}} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_i}$, $\theta^\mu \leftarrow \theta^\mu + \beta\nabla_{\theta^\mu} J$
     - Обновляем целевые сети
       $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
       $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
     - Уменьшаем шум $\mathcal{N}$

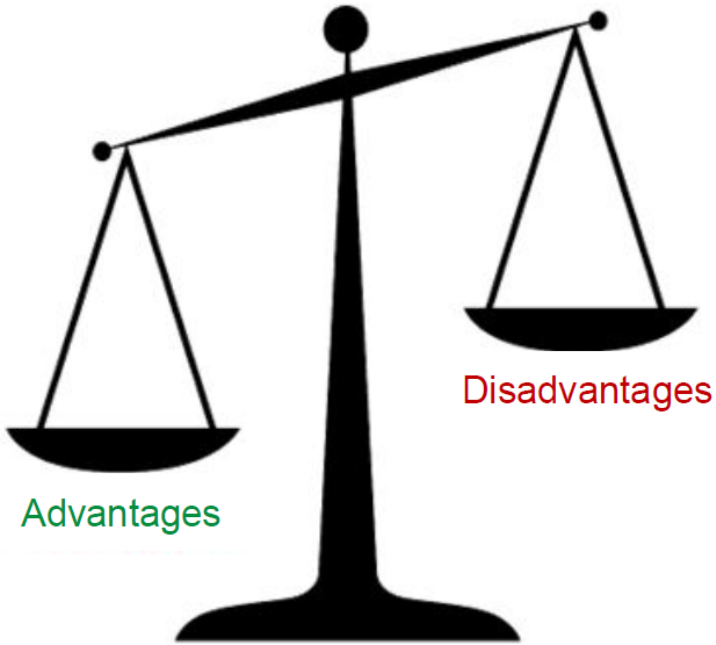
DDPG – достоинства и недостатки

Непрерывное пространство действий

Off-policy
Experience replay

Стабильнее обучение за счет использования суперпозиции
$Q(s, \pi^*(s))$

Реализация проще чем A3C



Работает **только** с непрерывным пространством действий

Практика

Практика

1. Реализовать алгоритм A2C
2. Реализовать алгоритм DDPG

Список материалов для изучения

1. Awesome Reinforcement Learning
   https://github.com/aikorea/awesome-rl

2. Memory-based Deep Reinforcement Learning for POMDPs
   https://arxiv.org/pdf/2102.12344

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Заполните, пожалуйста,
опрос о занятии
по ссылке в чате

Спасибо за внимание!