Обучение с подкреплением

Distributed RL

Для чего нужен Distributed RL?

Все рассмотренные ранее алгоритмы однопоточные, однако:

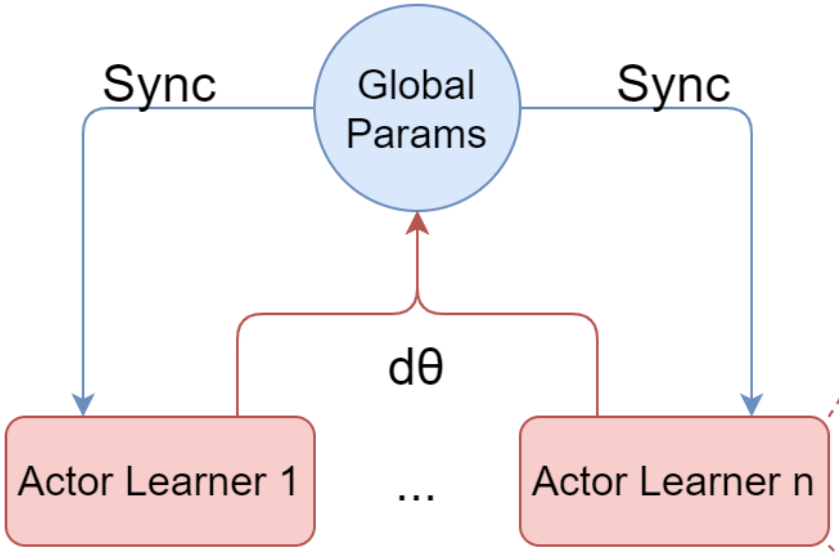
Если среда виртуальная:

1) Каждый компьютер, как правило, имеет несколько ядер CPU и может потенциально симулировать несколько сред параллельно
2) Часто в нашем распоряжении есть кластер из нескольких машин

Если среда реальная:

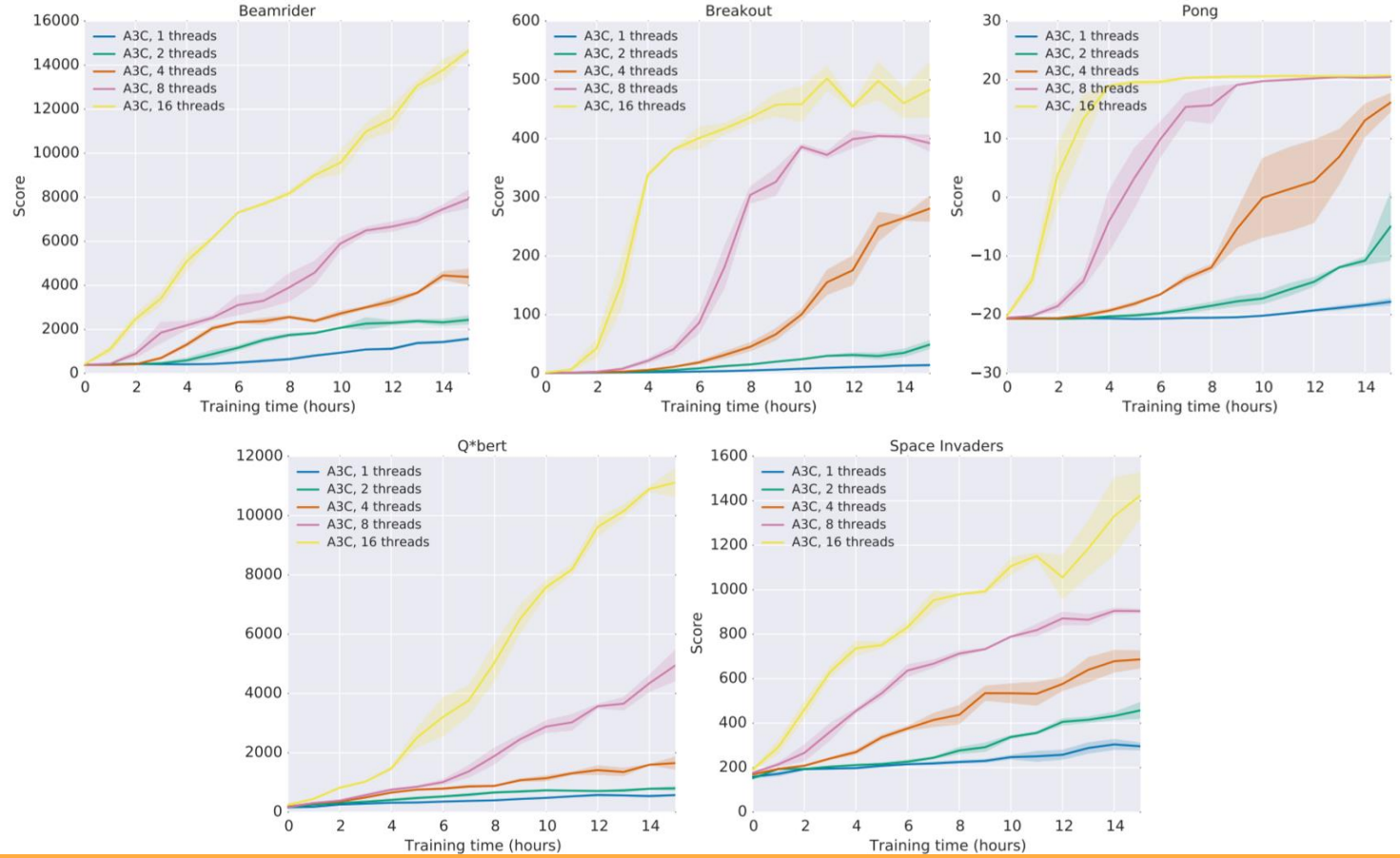
1) Можем создать несколько независимых копий среды (например, несколько одинаковых роботов)
2) Каждая среда очень медленная, поэтому получение опыта с использованием только одной среды займет вечность

Asynchronous Advantage Actor-Critic



1. Синхронизировать локальные веса θ' с глобальными весами веса θ
2. В течении T шагов или пока эпизод не завершился:
   2.1. Совершить действие $\mathbf{a} \sim \pi_{\theta'}$
   2.2. Получить награду **r** и новое состояние **s**
3. Посчитать оценку суммарной награды $\mathbf{R}_t$ для каждого шага траектории
4. Посчитать градиент $d\theta'$ с помощью **R**
5. $d\theta \leftarrow d\theta'$

Asynchronous Advantage Actor-Critic



Asynchronous Advantage Actor-Critic

Ограничения:

1) Веса модели централизованы. При большом количестве параллельных worker’ов будет слишком большая конкуренция за доступ к ним

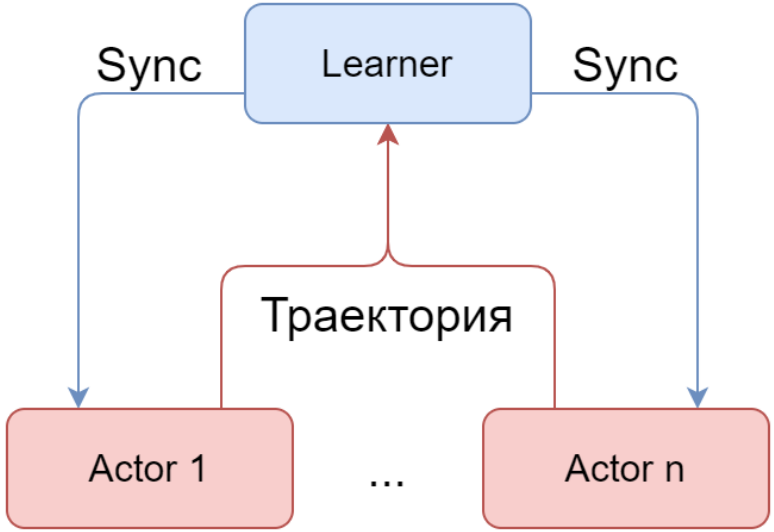
1) Версии моделей в worker’ах могут отличаться. С одной стороны, это позволяет улучшить исследование среды. С другой стороны, если отличие будет слишком сильным, то полученные worker’ом градиенты нельзя будет использовать.

1) Предполагается использование алгоритма на одном физическом устройстве, поскольку иначе накладные расходы на транспортировку параметров сетей и градиентов будут слишком велики.

IMPALA

Идея:
Будем передавать не градиенты, а траектории.

Проблема:
Для оптимизации используем on-policy алгоритм.
Передаваемые траектории инвалидируются.



IMPALA: V-Trace

Исправим проблему с устаревшей политикой, поменяв способ оценки суммарной награды:

$$v_t = V_\pi(s_t) + \sum_{i=0}^{T} \gamma^i \cdot (\prod_{k=0}^{i-1} c_{k+t}) \Delta_{i+t} V$$

где:

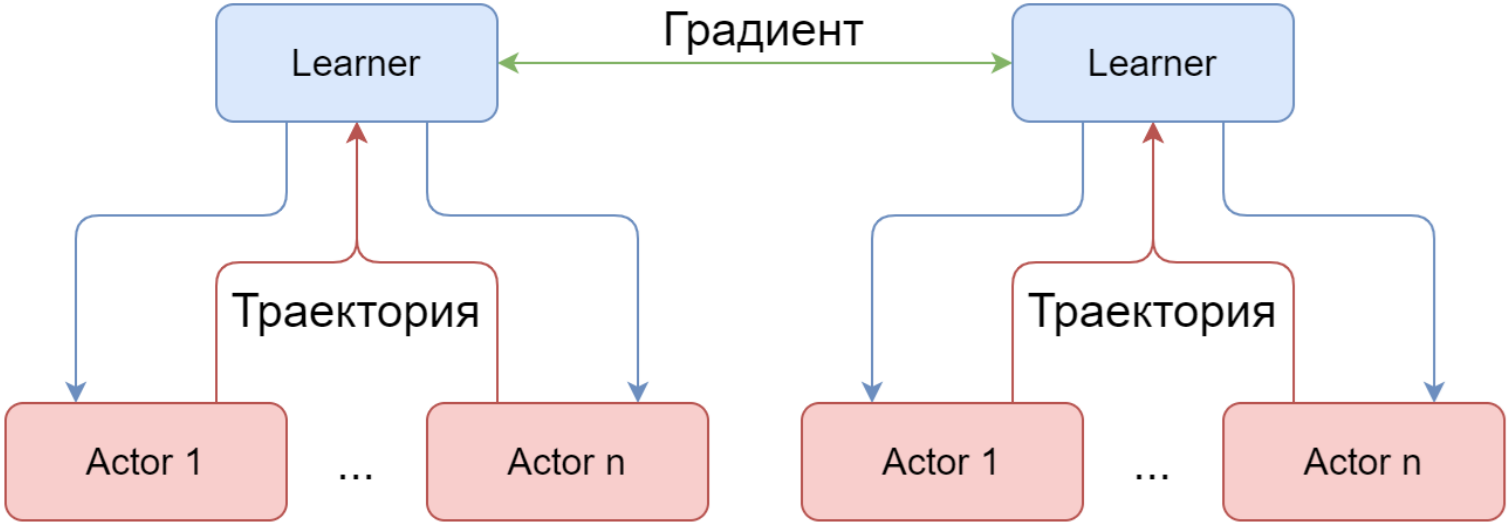
$$\Delta_{i+t} V = \text{clip}(\frac{\pi(a_{i+t} + s_{i+t})}{\mu(a_{i+t} + s_{i+t})}, 0, \bar{p})(r_{i+t} + \gamma V_\pi(s_{i+t+1}) - V_\pi(s_{i+t}))$$

$$c_{k+t} = \text{clip}(\frac{\pi(a_{i+t} + s_{i+t})}{\mu(a_{i+t} + s_{i+t})}, 0, \bar{c})$$

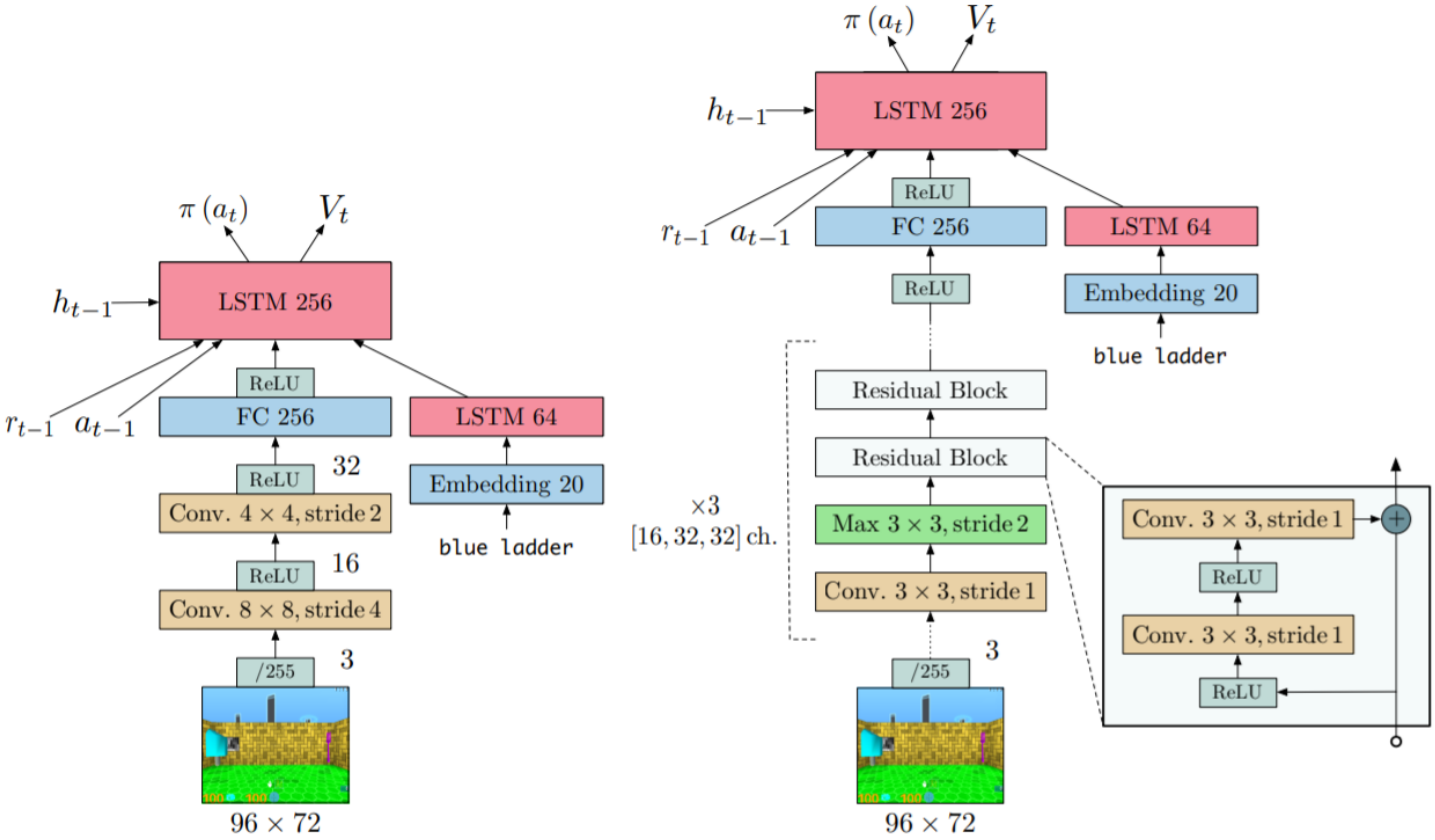
$$\bar{p} \geq \bar{c} \geq 1$$

IMPALA

А еще можем параллелить несколько Learner’ов



IMPALA: архитектура сети



IMPALA

Замечания:

1. Алгоритм использует A2C для обучения политики
1. V-Trace можно обобщить, если использовать $c_i^\lambda = \lambda \cdot c_i$
1. В передаваемые траектории также записана информация и о рапределении действий политики (необходима для подсчета V-trace)
1. IMPALA более масштабируема, чем A3C
1. Узким местом может быть обмен градиентами между learner'ами, особенно, если их много

Напоминание: Prioritized Experience Replay

Идея: Будем чаще обучаться на том опыте, на котором больше ошибка

Берем опыт из буфера с вероятностью:

$$p((s, a, s', r, d)_i) = \frac{p_i^\alpha}{\sum p_j^\alpha}$$

А затем корректируем изменение распределения с помощью взвешенного обновления нейронной сети:

Importance Sampling Сглаживание

$$w_i = \frac{[N \cdot p(s, a, s', r, d)_i))]^{-\beta}}{\max_j w_j}$$

Нормализация

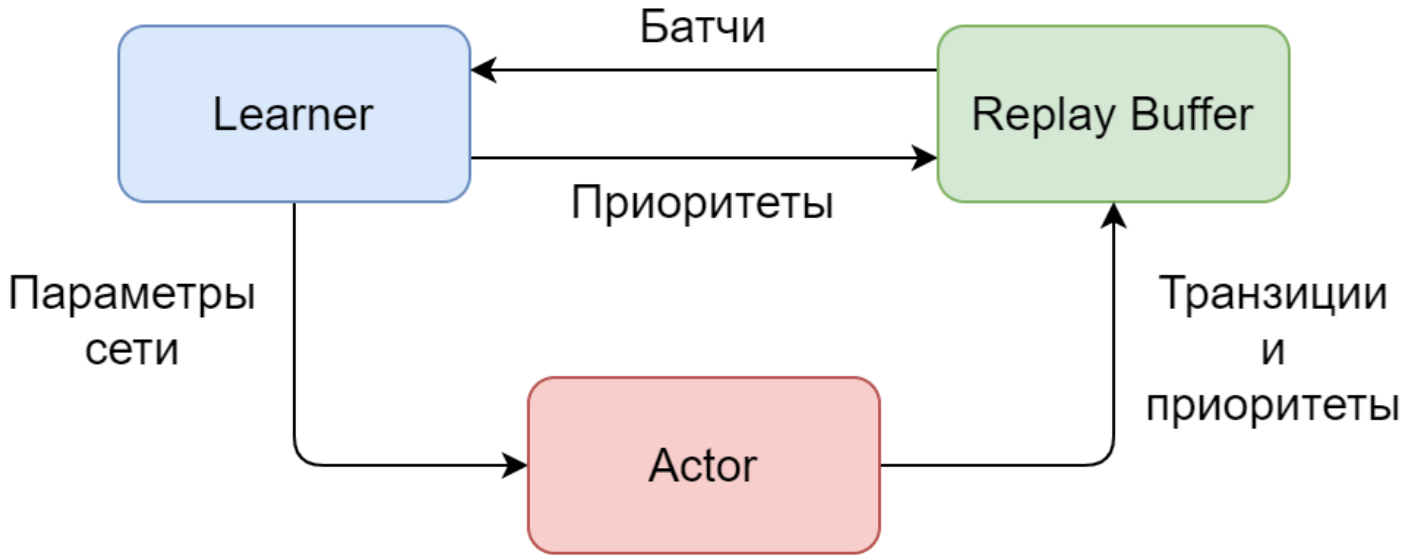
Ape-X

IMPALA и A3C могут использоваться для обучения off-policy алгоритмов, однако они не предоставляют возможности использовать буфер опыта.

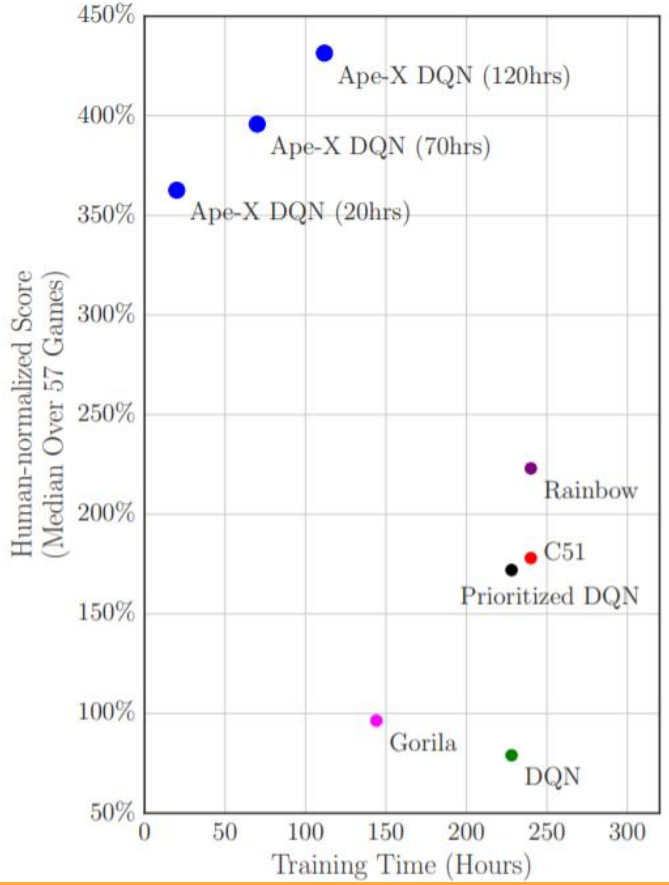
Идея:
Добавим буфер опыта, в который actor’ы будут складывать данные.

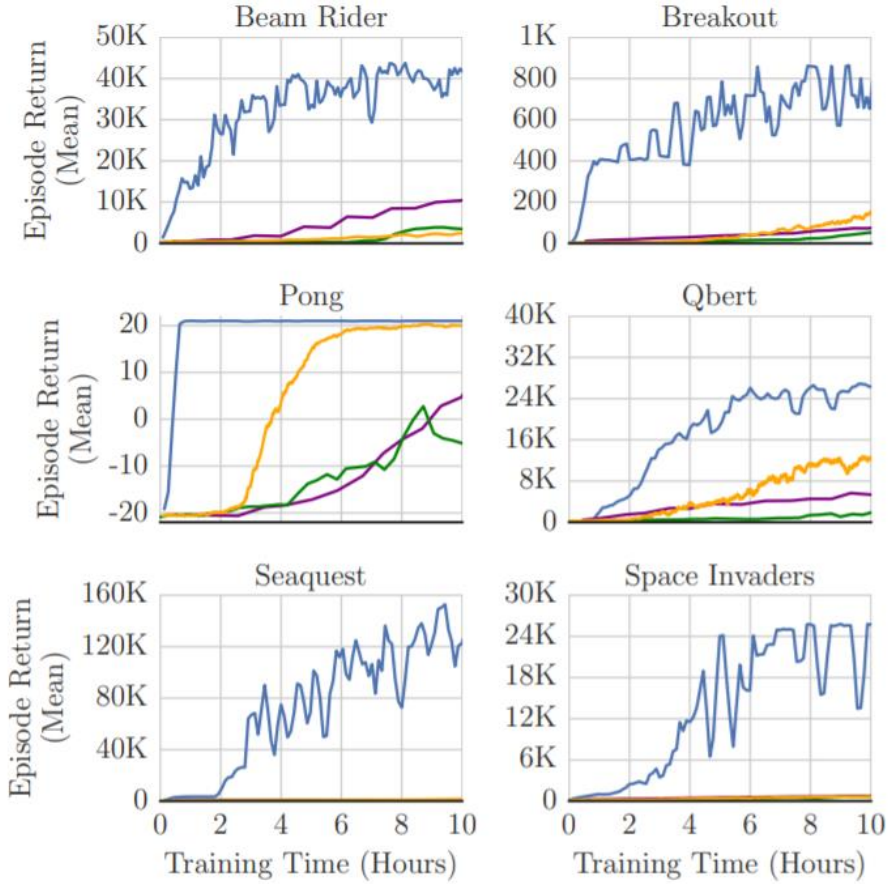
Замечание:
Опыт может меняться очень быстро, поэтому его важно приоритизировать.

Ape-X


Батчи
Learner
Replay Buffer
Приоритеты
Параметры
сети
Транзиции
и
приоритеты
Actor

Ape-X


Human-normalized Score (Median Over 57 Games)
450%
400%
350%
300%
250%
200%
150%
100%
50%
0
50
100
150
200
250
300
Training Time (Hours)
Ape-X DQN (120hrs)
Ape-X DQN (70hrs)
Ape-X DQN (20hrs)
Rainbow
C51
Prioritized DQN
Gorila
DQN


Beam Rider
Breakout
Pong
Qbert
Seaquest
Space Invaders
Episode Return (Mean)
Training Time (Hours)

R2D2

Что делать, если хотим использовать RNN как в IMPALA?

Проблема: Классический приоритезированный буфер опыта не позволяет хранить куски траекторий

Решение: Будем хранить не отдельные транзиции, а части траекторий. Нужно научиться считать приоритеты для последовательностей.

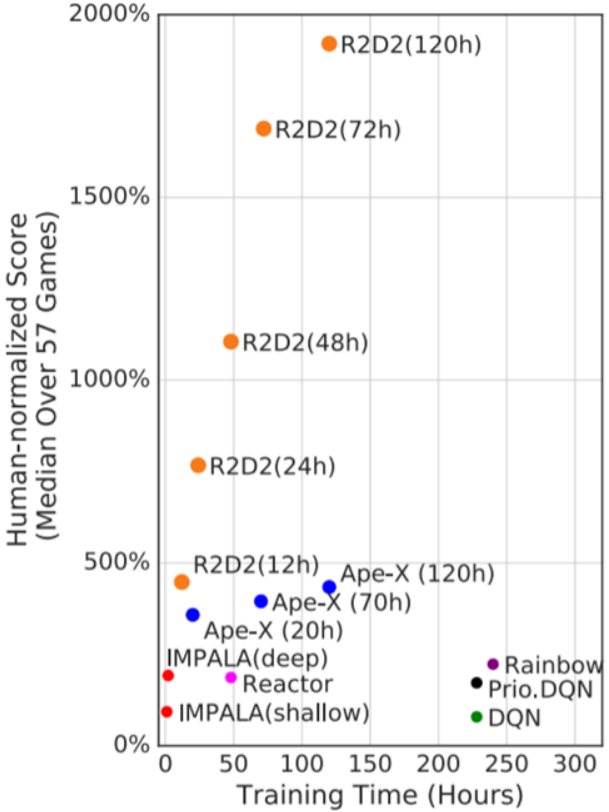
R2D2

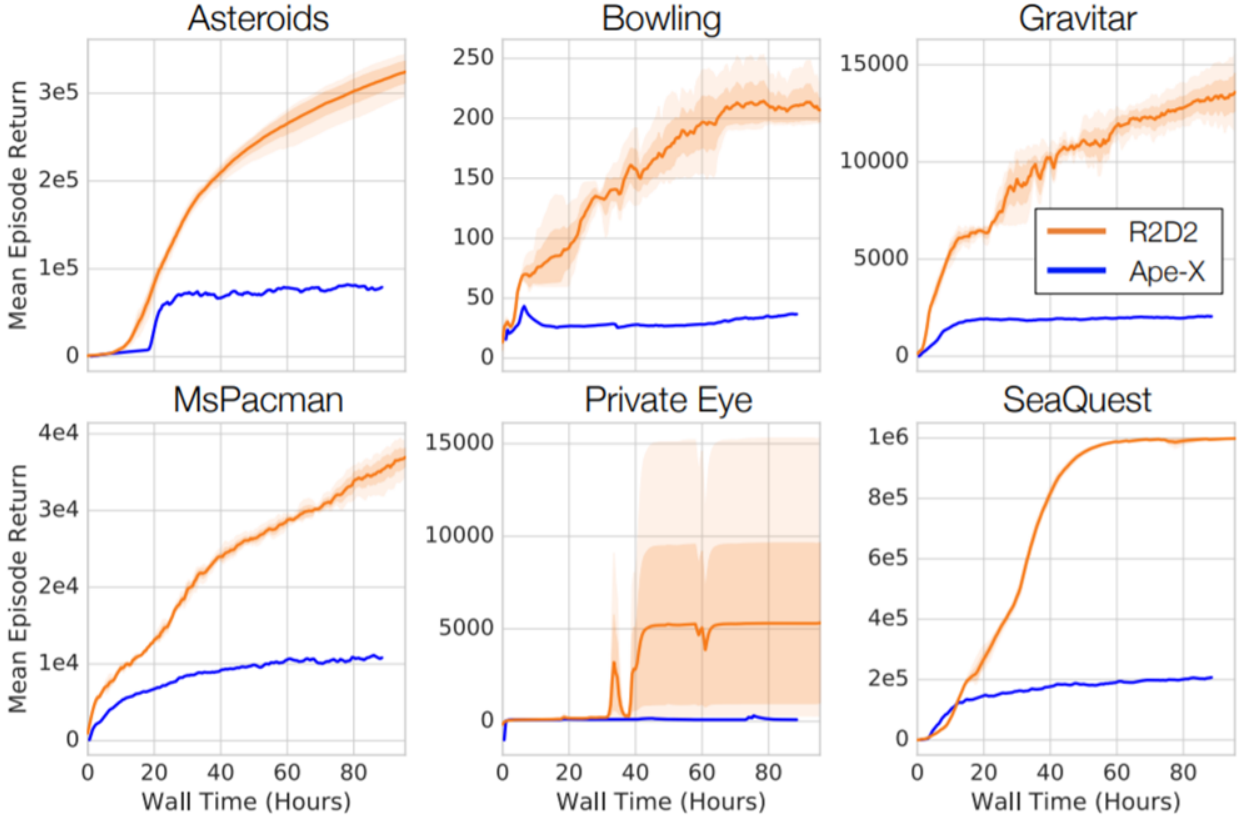
Пусть δ_i - ошибка предсказания для перехода из последовательности. Тогда приоритет определим как:

$$p = \eta \max_i \delta_i + (1 - \eta)\text{mean } \delta_i$$

Т.е., балансируем между максимальной и средней ошибками.

R2D2


2000%
1500%
1000%
500%
0%
Human-normalized Score
(Median Over 57 Games)
R2D2(120h)
R2D2(72h)
R2D2(48h)
R2D2(24h)
R2D2(12h)
Ape-X (120h)
Ape-X (70h)
Ape-X (20h)
IMPALA(deep)
Reactor
IMPALA(shallow)
Rainbow
Prio.DQN
DQN
0
50
100
150
200
250
300
Training Time (Hours)


Asteroids
Bowling
Gravitar
MsPacman
Private Eye
SeaQuest
Mean Episode Return
Wall Time (Hours)
R2D2
Ape-X