Обучение с подкреплением

Offline RL

Напоминание: Обучение по демонстрациям

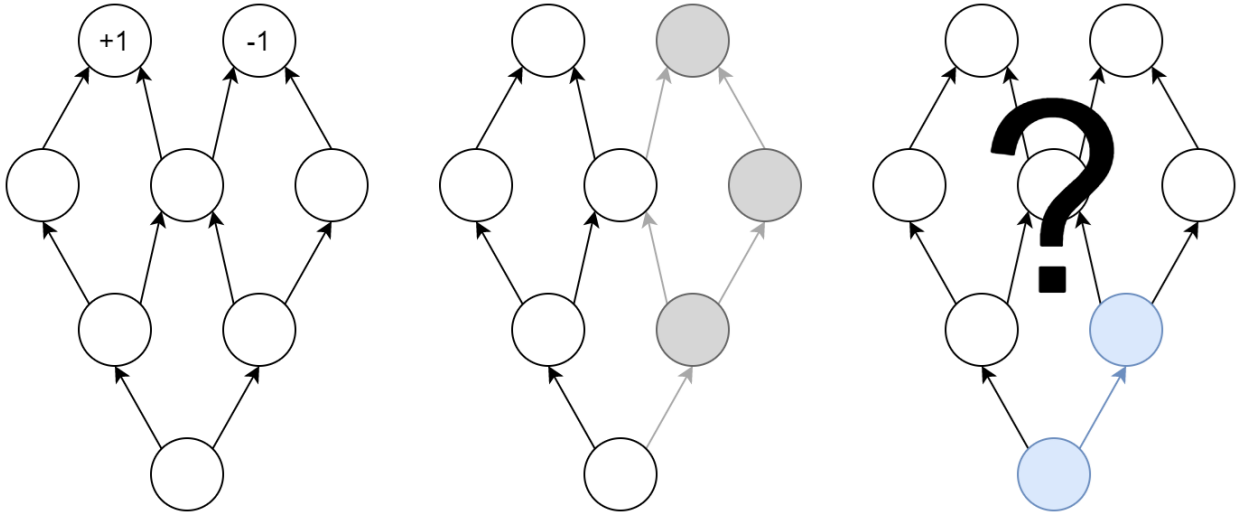
Предположим, что существует эксперт, который умеет решать задачу и может продемонстрировать несколько решений

Более формально:
Есть набор данных, содержащий траектории из окружения, полученные благодаря эксперту. Каждая траектория - последовательность переходов в среде, включающая состояние, действие и награду.

Напоминание: Behavioral Cloning

Наивное решение:
Используем любой алгоритм классификации/регрессии для того, чтобы как можно лучше приблизить действия эксперта по набору данных

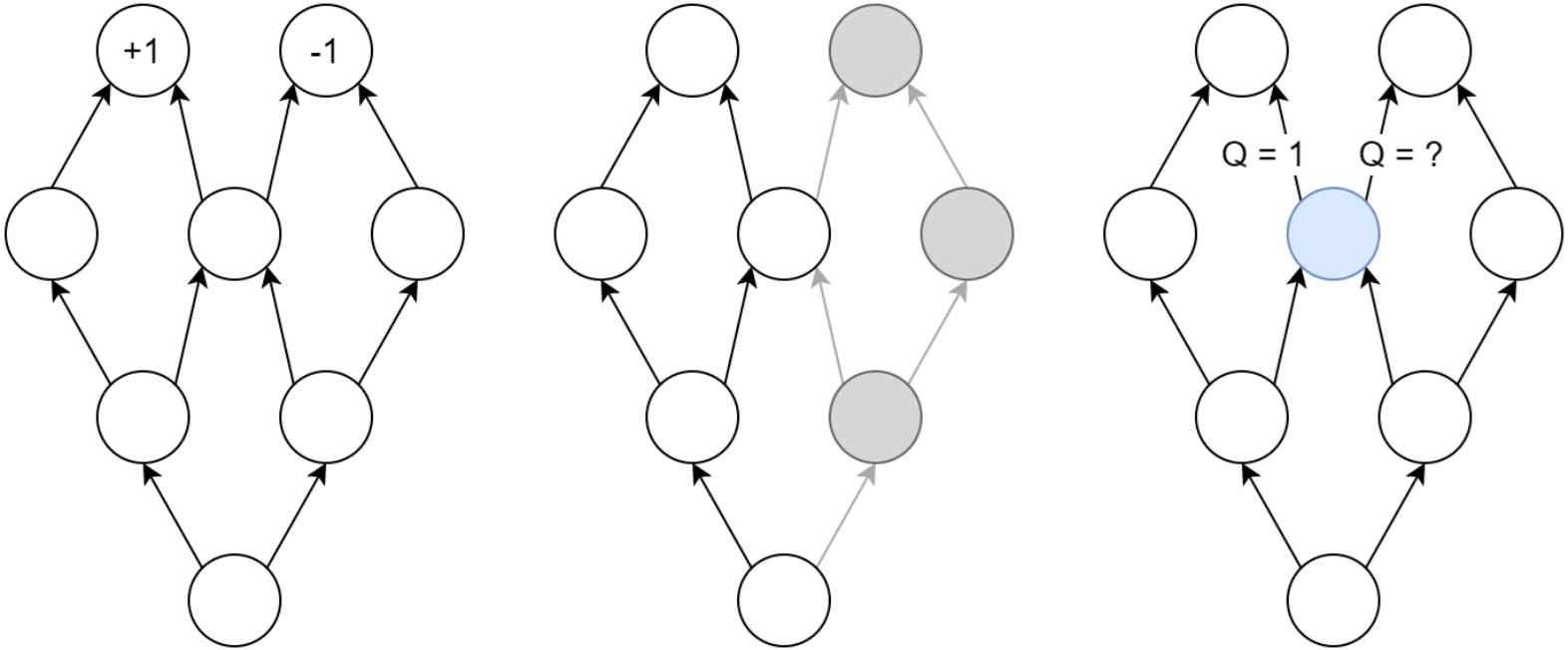
Проблема:



Напоминание: DQfD

Идея: будем обучать DQN по данным из демонстраций эксперта

Проблема:



Offline Reinforcement Learning

Задача Offline RL

Цель формулируется также, как и в случае обучения с подкреплением: максимизировать суммарную дисконтированную награду

Однако теперь предполагаем, что нет возможности взаимодействовать с окружением, но зато есть набор данных, которые можем использовать для обучения

Также теперь считаем, что данные могут быть субоптимальными, т.е. недостаточно просто скопировать поведение эксперта

Подходы к Offline RL

Основные подходы делятся на три типа:

- Importance Sampling в Policy Gradient методах
  - Importance Sampling часто приводит к неточной оценке

- Off-policy Reinforcement Learning
  - Также, как и в случае с DQfD, присутствует проблема ошибки в оценке Q-function для пар (состояние, действие), которых нет в датасете

- Model-based Reinforcement Learning
  - Ошибки в построении модели могут привести к тому, что оптимальная политика с точки зрения модели не будет оптимальной в реальной среде
  - Для пар (состояние, действие), которых нет в датасете, можем также неверно предсказывать награду или следующее состояние

Policy Constraints

Идея: если итоговая политика не сильно отличается от экспертной политики, то ошибка будет небольшой

В частности, предлагается решать задачу:

$$\pi = \arg\max_{\pi} Q_{\pi}(s,a) \text{ s.t. } D_{\mathrm{KL}}(\pi, \pi_{\mathrm{expert}}) \leq \epsilon$$

Подойти к ее решению можно двумя способами:

1. Использовать регуляризацию дивергенции вместо жесткого ограничения

1. Аналитически решить задачу нахождения оптимальной политики для текущего приближения Q с учетом KL-дивергенции:

$$\hat{\pi} = \frac{1}{\epsilon}\pi_{\mathrm{expert}}(a|s)\exp(Q_{\pi}(s,a))$$

А затем найти такую политику, что $D_{\mathrm{KL}}(\pi, \hat{\pi}) \to \min$

Conservative Q-learning

Идея: “Раньше было лучше”

Поскольку основная проблема политики заключается в переоценке Q-function для действий не из набора данных, регуляризуем Q-function. Например, так:

$$\phi = \mathbb{E}_s \mathbb{E}_{a \sim \mu} Q_\pi(s, a)$$

Где $\mu = \arg\max \mathbb{E}[Q_\pi(s, a) + \mathcal{H}(\mu)(.|s)]$

Причем, нам даже не нужно обучать adversarial политику, т.к. есть аналитическое решение:

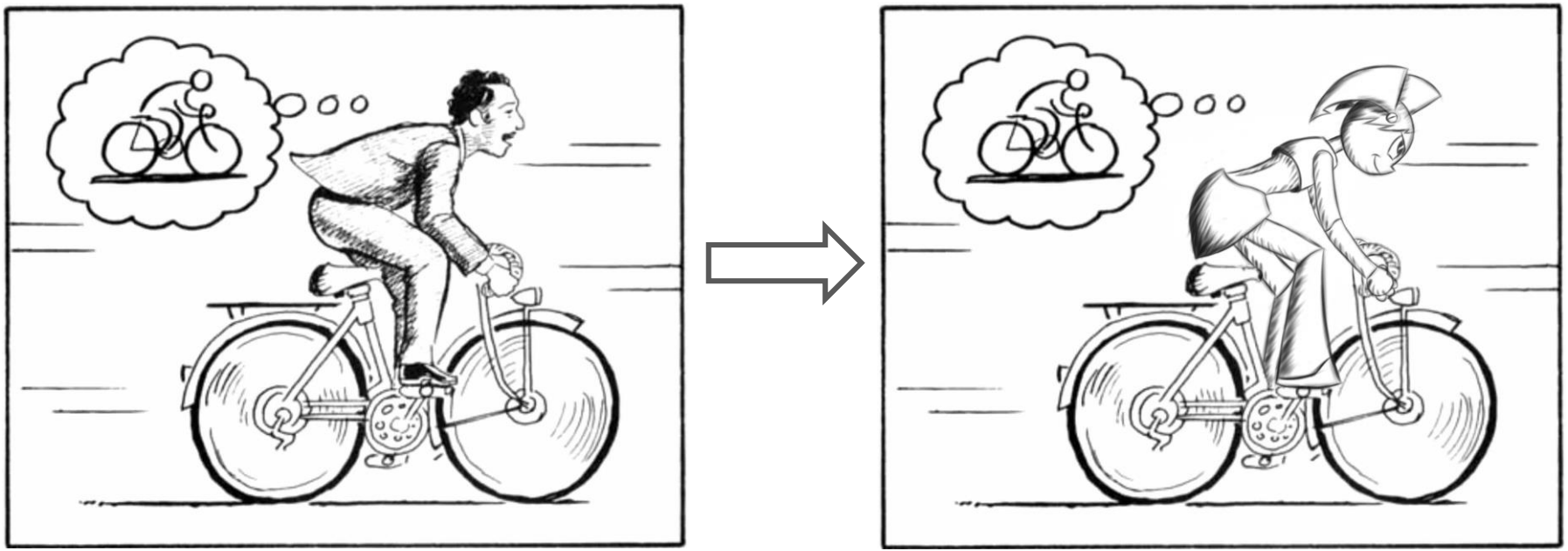
$$\phi = \mathbb{E}_s \log \sum_a \exp Q_\pi(s, a)$$

Однако, чтобы избежать излишне консервативной функции (что приведет к lower bound оценке), мы можем преобразовать регуляризацию следующим образом:

$$\phi = \mathbb{E}_s [\mathbb{E}_{a \sim \mu} Q_\pi(s, a) - \mathbb{E}_{a \sim \pi_{\text{expert}}} Q_\pi(s, a)]$$

Model-based Reinforcement Learning

Model-based RL



World Model

At each time step, our agent receives an **observation** from the environment.

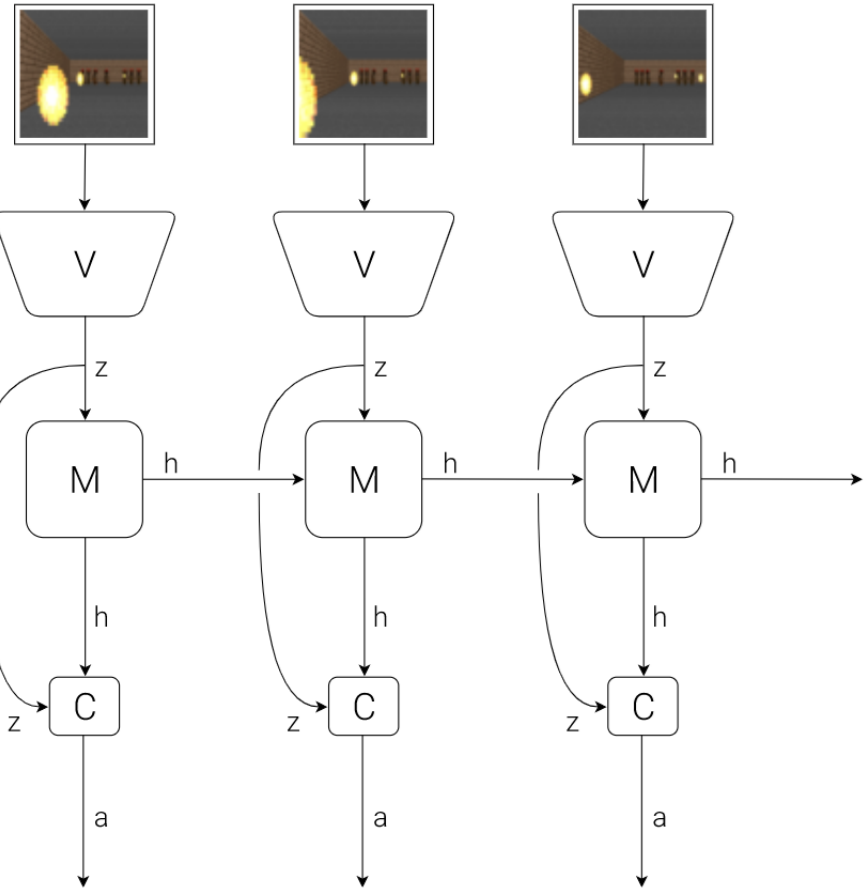
World Model

The **Vision Model (V)** encodes the high-dimensional observation into a low-dimensional latent vector.

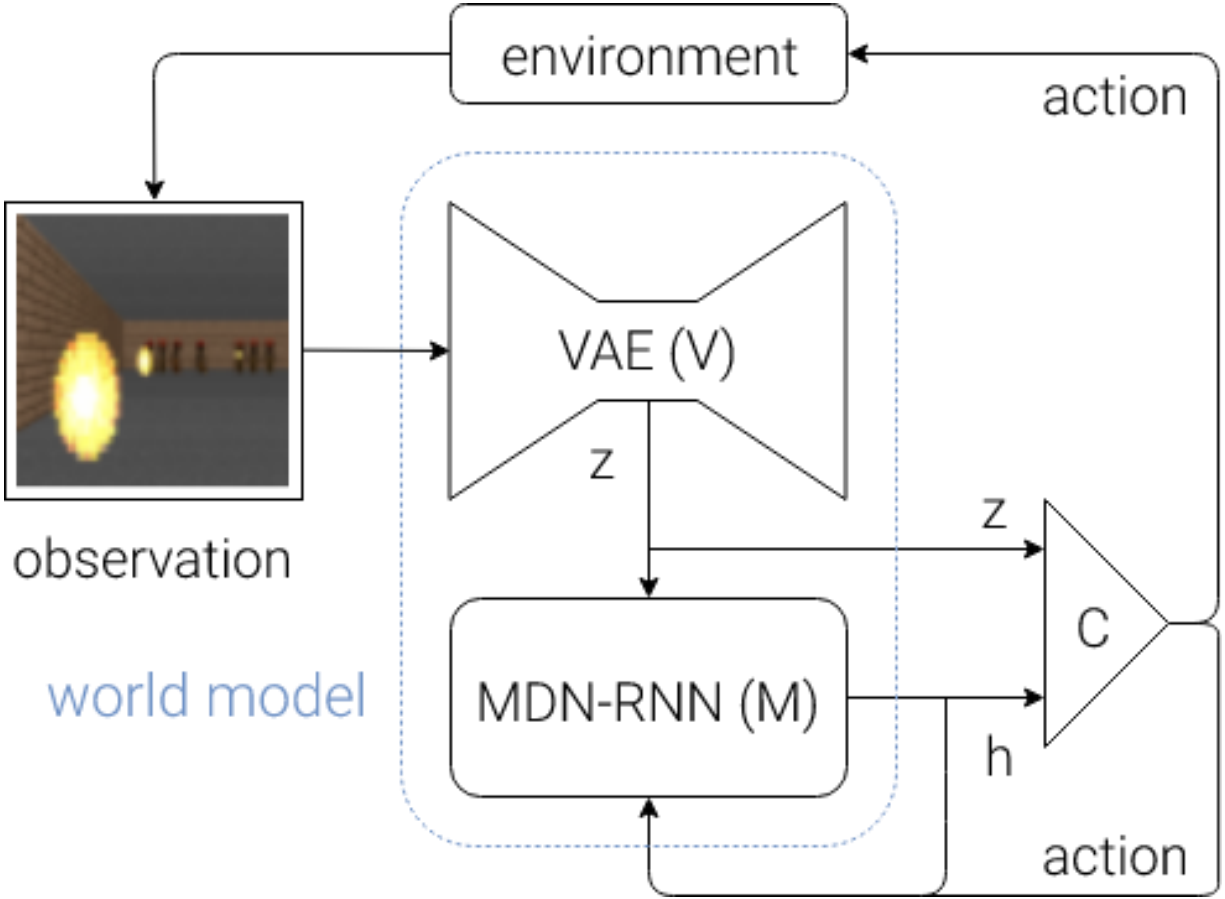
The **Memory RNN (M)** integrates the historical codes to create a representation that can predict future states.

A small **Controller (C)** uses the representations from both **V** and **M** to select good actions.

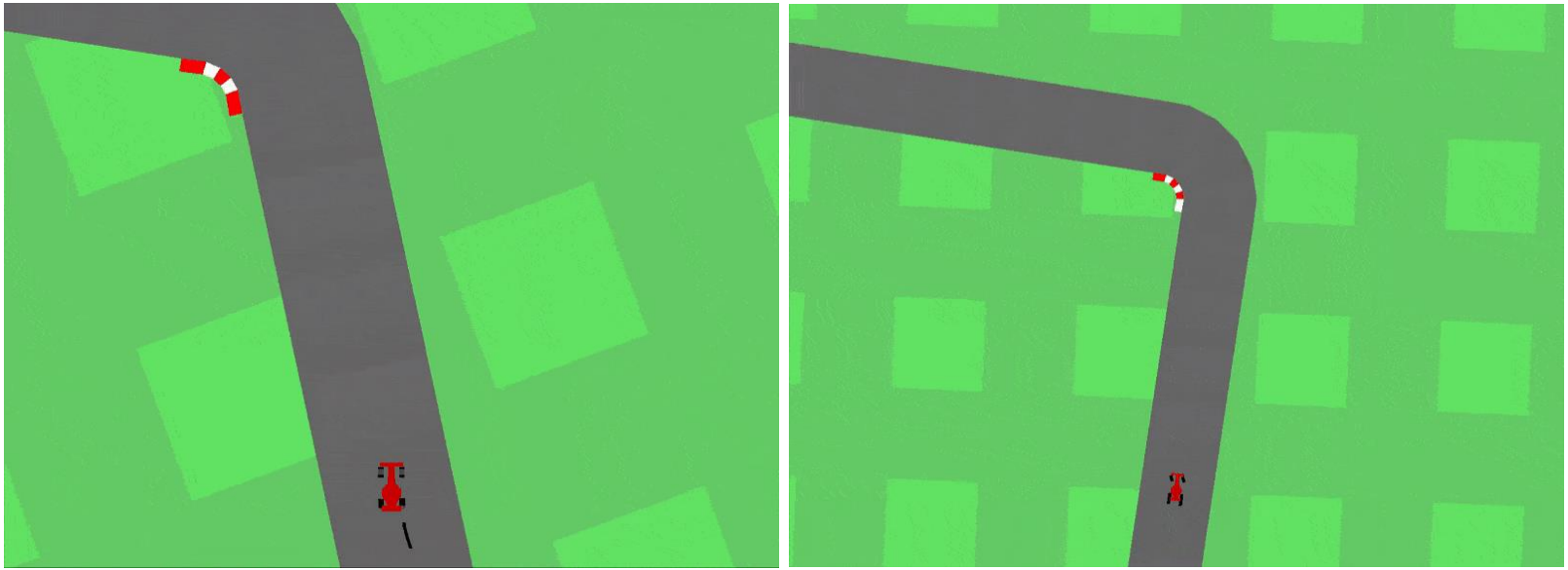
The agent performs **actions** that go back and affect the environment.



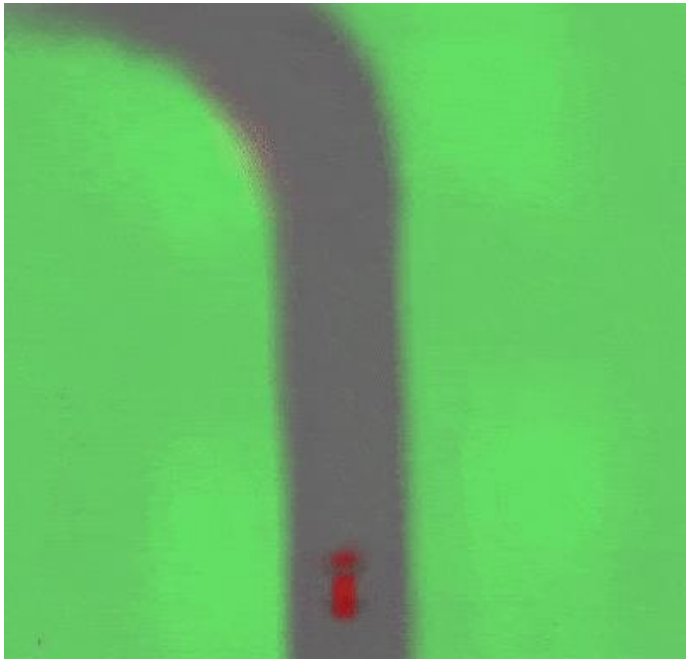
World Model


environment
action
VAE (V)
z
observation
z
C
world model
MDN-RNN (M)
h
action

World Model: Car Racing



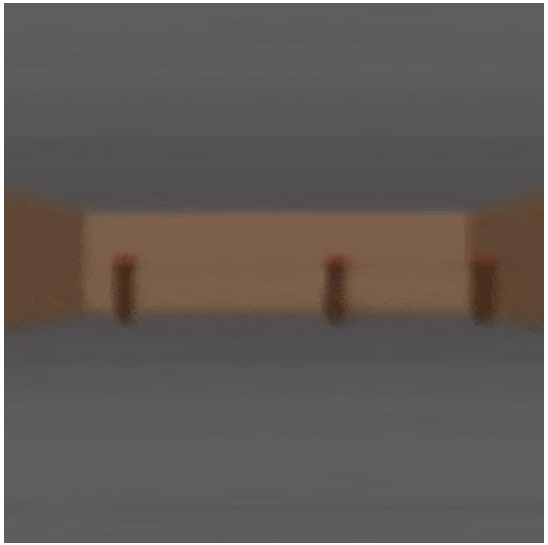
World Model: Car Racing



World Model: VizDoom

Temperature	Score in Virtual Environment	Score in Actual Environment
0.10	2086 ± 140	193 ± 58
0.50	2060 ± 277	196 ± 50
1.00	1145 ± 690	868 ± 511
1.15	918 ± 546	1092 ± 556
1.30	732 ± 269	753 ± 139
Random Policy Baseline	N/A	210 ± 108
Gym Leaderboard [34]	N/A	820 ± 58

World Model: VizDoom



PlaNet

Перейдем к задаче POMDP (т.е., агент не видит полное состояние среды)
Дискретное время t, состояние s_t, действие a_t, наблюдение o_t, награда r_t

Тогда:

Стратегия агента - $a_t \sim p(a_t | o_{\leq t}, a_{<t})$
Функция награды - $r_t \sim p(r_t | s_t)$
Наблюдение - $o_t \sim p(o_t | s_t)$
Функция перехода - $s_t \sim p(s_t | s_{t-1}, a_{t-1})$

PlaNet

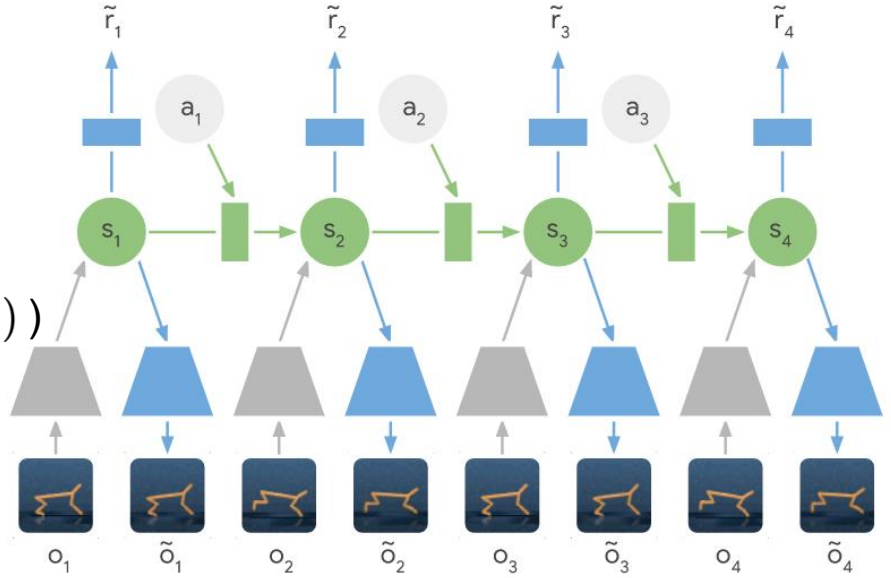
Что учит PlaNet:

$p(s_t|s_{t-1}, a_{t-1})$

$p(o_t|s_t)$

$p(r_t|s_t)$

$q(s_t|o_{\leq t}, a_{<t})$ (на самом деле $q(s_t|s_{t-1}, o_t, a_t)$)



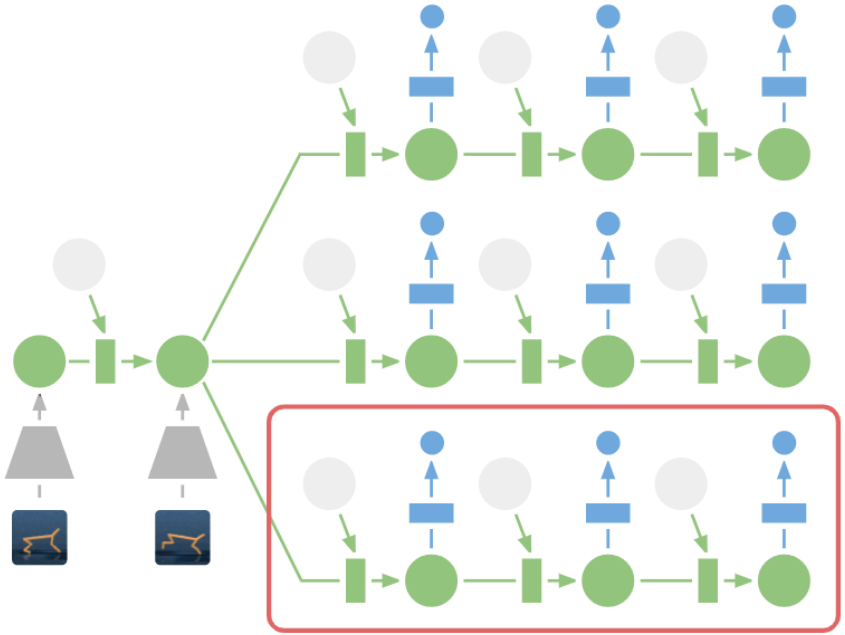
Важно: состояние здесь не обязательно такое же, как на самом деле в среде

Данные для обучения собираются итеративно.

PlaNet

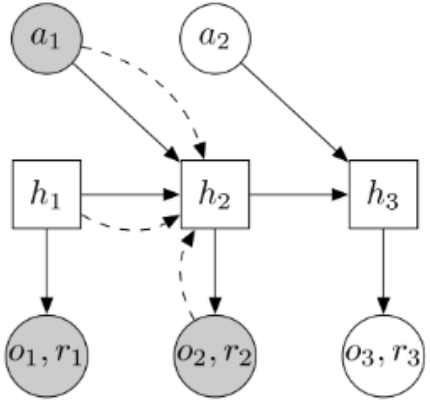
Используем MPC (популяционный подход) для планирования траектории с использованием CEM:

1. Инициализировать распределение стандартным нормальным распределением
2. Повторяем в течении I итераций:
   3. Получить K последовательностей действий из распределения $a_{t:t+T} \sim N(\mu_{t:t+T}, \sigma^2_{t:t+T})$

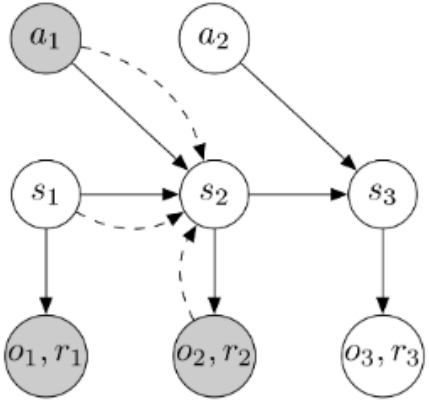


   1. Выбрать лучшую последовательность, максимизировать ее вероятность

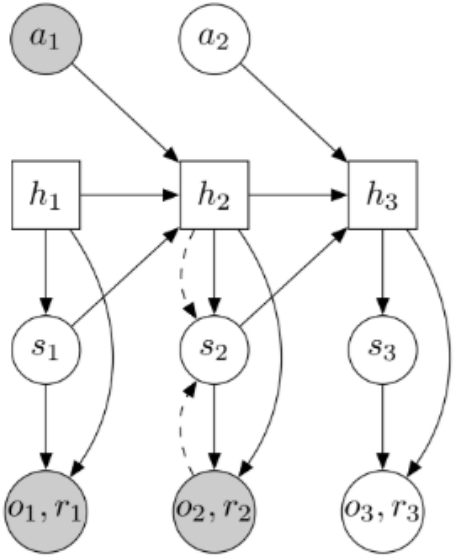
PlaNet: RSSM



(a) Deterministic model (RNN)

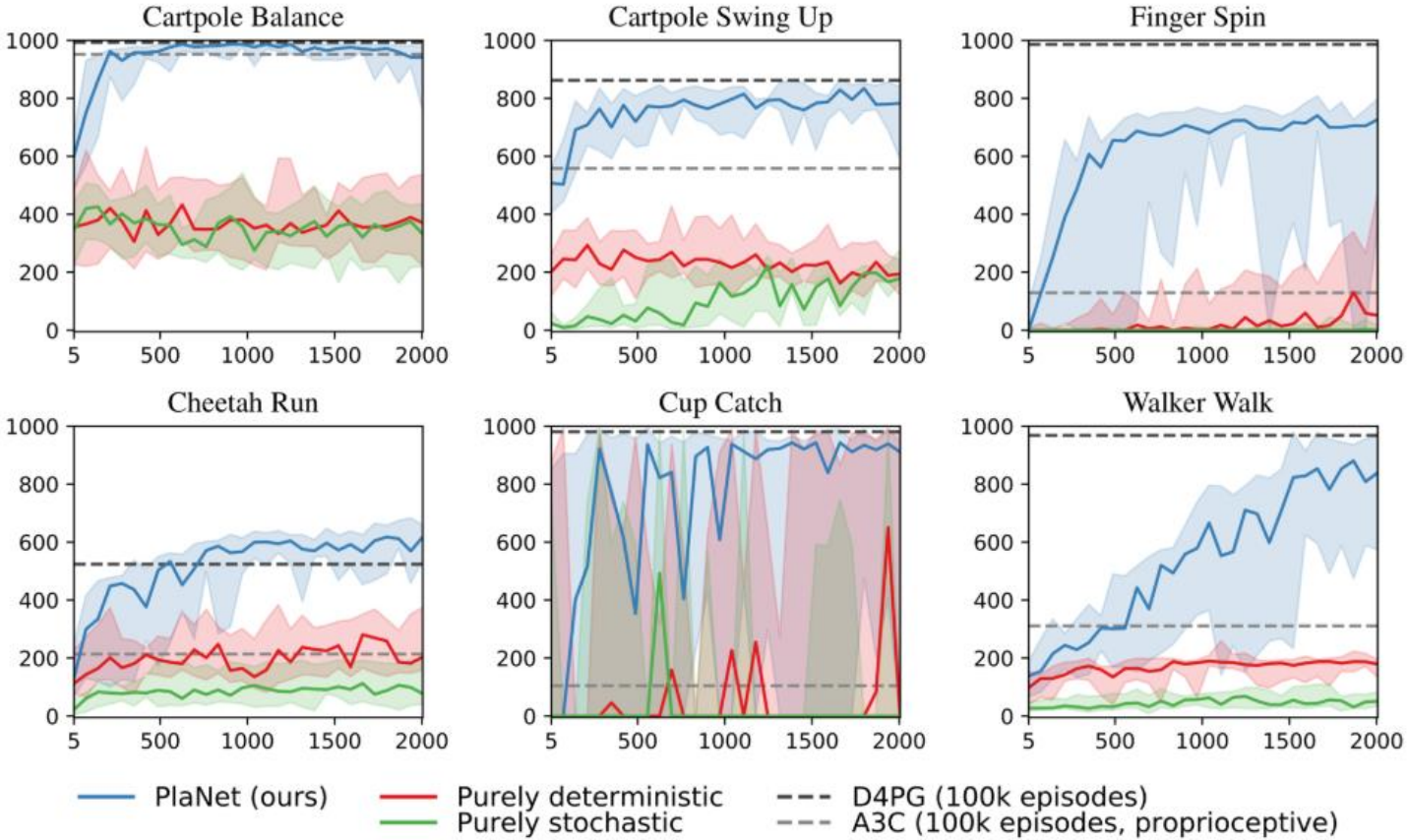


(b) Stochastic model (SSM)

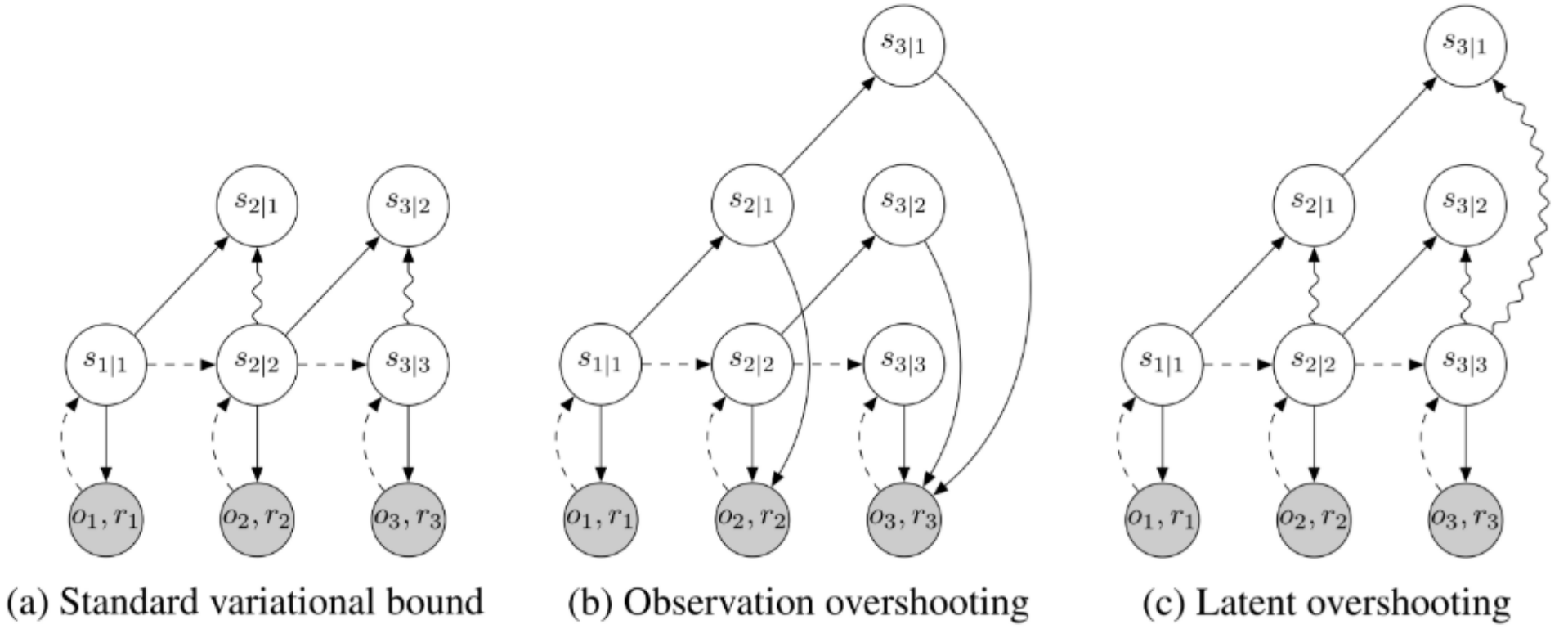


(c) Recurrent state-space model (RSSM)

PlaNet: RSSM


Cartpole Balance
Cartpole Swing Up
Finger Spin
Cheetah Run
Cup Catch
Walker Walk
0
200
400
600
800
1000
5
500
1000
1500
2000
PlaNet (ours)
Purely deterministic
Purely stochastic
D4PG (100k episodes)
A3C (100k episodes, proprioceptive)

PlaNet: Latent Overshooting



(a) Standard variational bound (b) Observation overshooting (c) Latent overshooting

PlaNet: Latent Overshooting

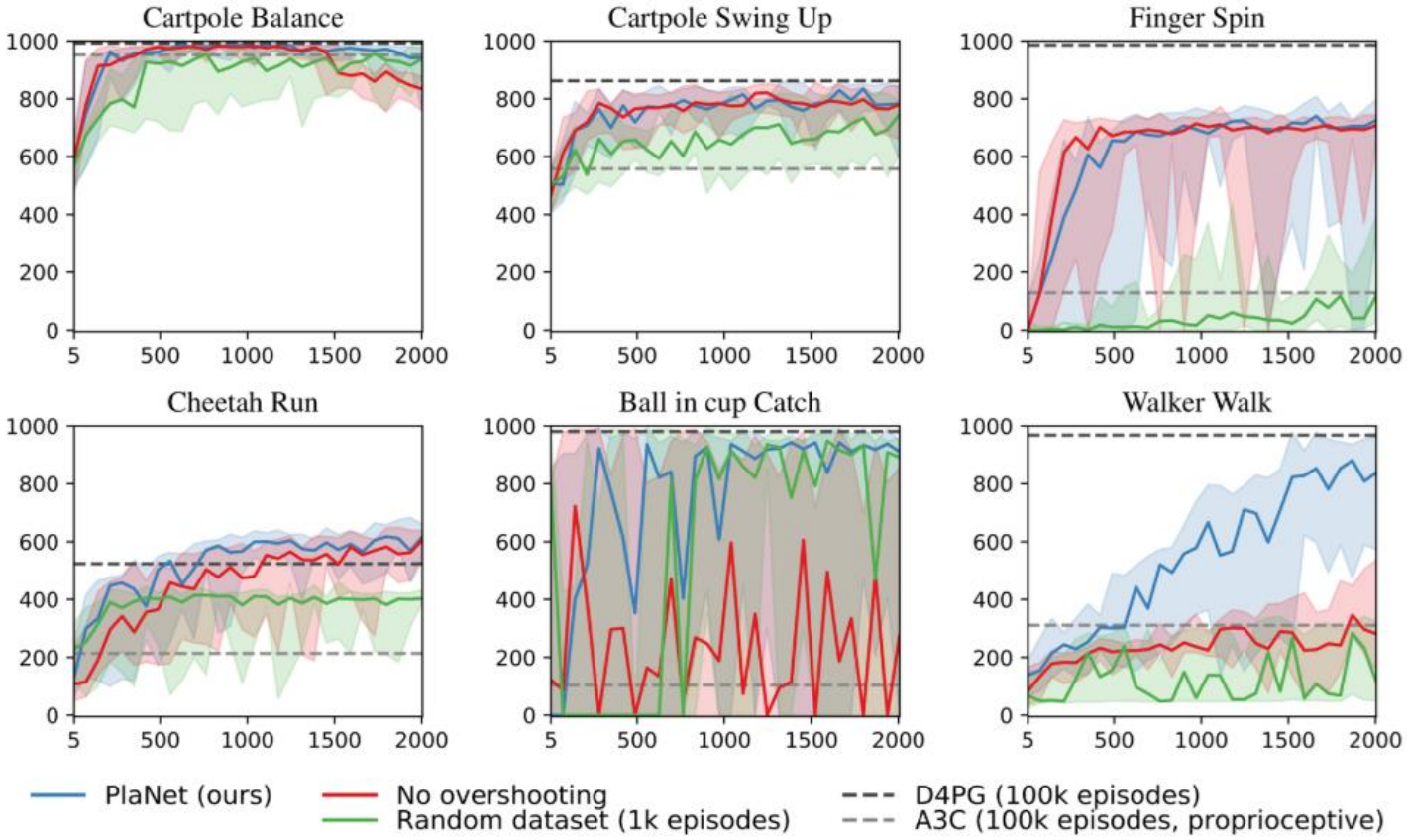
Обычная функция потерь:

$$\sum_{t=1}^{T} \Bigg(\underbrace{\mathrm{E}_{q(s_t|o_{\leqslant t},a_{<t})}[\ln p(o_t \mid s_t)]}_{\text{reconstruction}} \hookleftarrow - \underbrace{\mathrm{E}_{q(s_{t-1}|o_{\leqslant t-1},a_{<t-1})}\big[\mathrm{KL}[q(s_t \mid o_{\leqslant t}, a_{<t}) \parallel p(s_t \mid s_{t-1}, a_{t-1})]\big]}_{\text{complexity}} \Bigg)$$

Latent Overshooting:

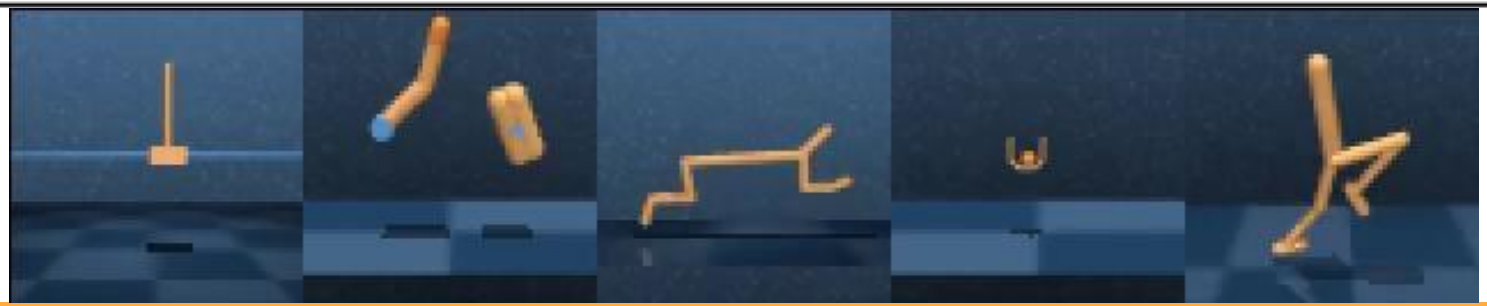
$$\sum_{t=1}^{T} \Bigg(\underbrace{\mathrm{E}_{q(s_t|o_{\leqslant t})}[\ln p(o_t \mid s_t)]}_{\text{reconstruction}} \hookleftarrow - \underbrace{\frac{1}{D}\sum_{d=1}^{D} \beta_d \, \mathrm{E}_{p(s_{t-1}|s_{t-d})q(s_{t-d}|o_{\leqslant t-d})}\big[\mathrm{KL}[q(s_t \mid o_{\leqslant t}) \parallel p(s_t \mid s_{t-1})]\big]}_{\text{latent overshooting}} \Bigg).$$

PlaNet: Latent Overshooting

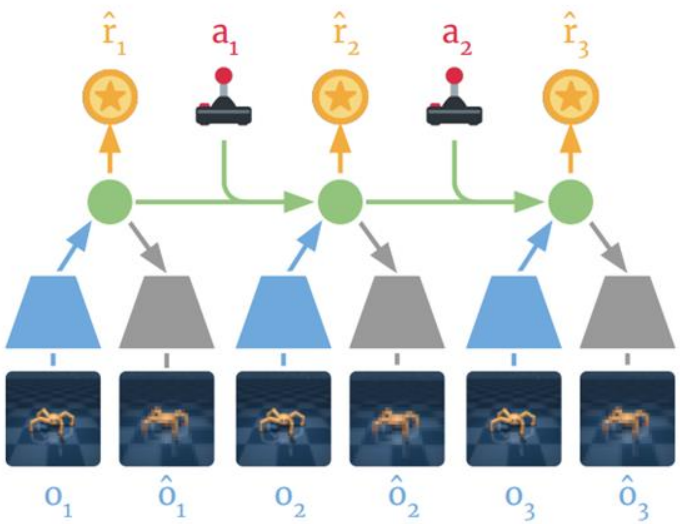


PlaNet: Results

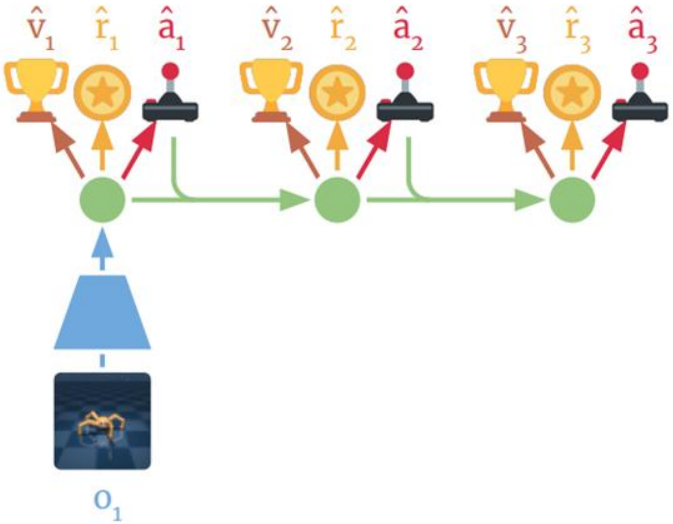
Method	Modality	Episodes	Cartpole Balance	Cartpole Swingup	Finger Spin	Cheetah Run	Ball in cup Catch	Walker Walk
A3C	proprioceptive	100,000	952	558	129	214	105	311
D4PG	pixels	100,000	993	862	985	524	980	968
PlaNet (ours)	pixels	2,000	986	831	744	650	914	890
CEM + true simulator	simulator state	0	998	850	825	656	993	994
Data efficiency gain PlaNet over D4PG (factor)			100	180	16	50+	20	11



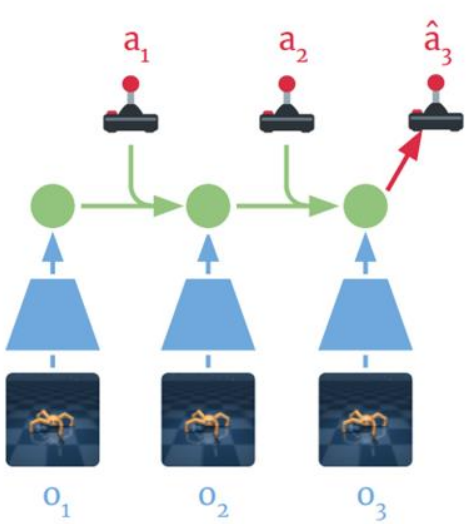
Dreamer



(a) Learn dynamics from experience



(b) Learn behavior in imagination



(c) Act in the environment

Dreamer

В Dreamer’е модель мира обучается так же, как в PlaNet, но вместо планирования используется агент

$$\mathrm{V}_{\mathrm{R}}(s_\tau) \doteq \mathrm{E}_{q_\theta, q_\phi}\left(\sum_{n=\tau}^{t+H} r_n\right), \tag{4}$$

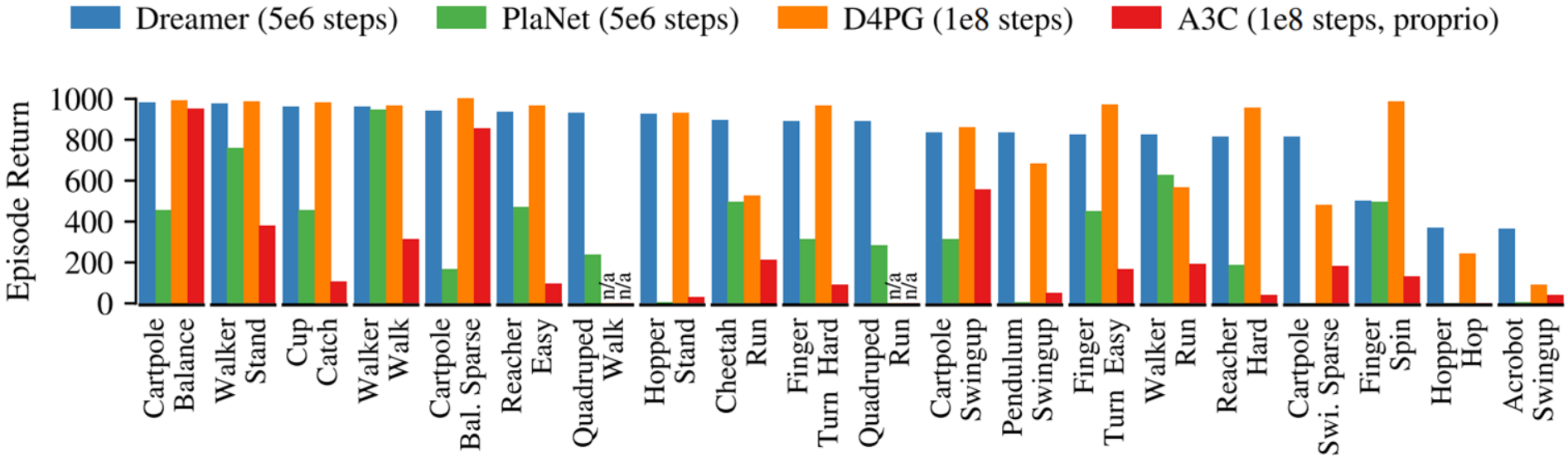
$$\mathrm{V}_{\mathrm{N}}^{k}(s_\tau) \doteq \mathrm{E}_{q_\theta, q_\phi}\left(\sum_{n=\tau}^{h-1} \gamma^{n-\tau} r_n + \gamma^{h-\tau} v_\psi(s_h)\right) \quad \text{with} \quad h = \min(\tau + k, t + H), \tag{5}$$

$$\mathrm{V}_\lambda(s_\tau) \doteq (1-\lambda) \sum_{n=1}^{H-1} \lambda^{n-1} \mathrm{V}_{\mathrm{N}}^{n}(s_\tau) + \lambda^{H-1} \mathrm{V}_{\mathrm{N}}^{H}(s_\tau), \tag{6}$$

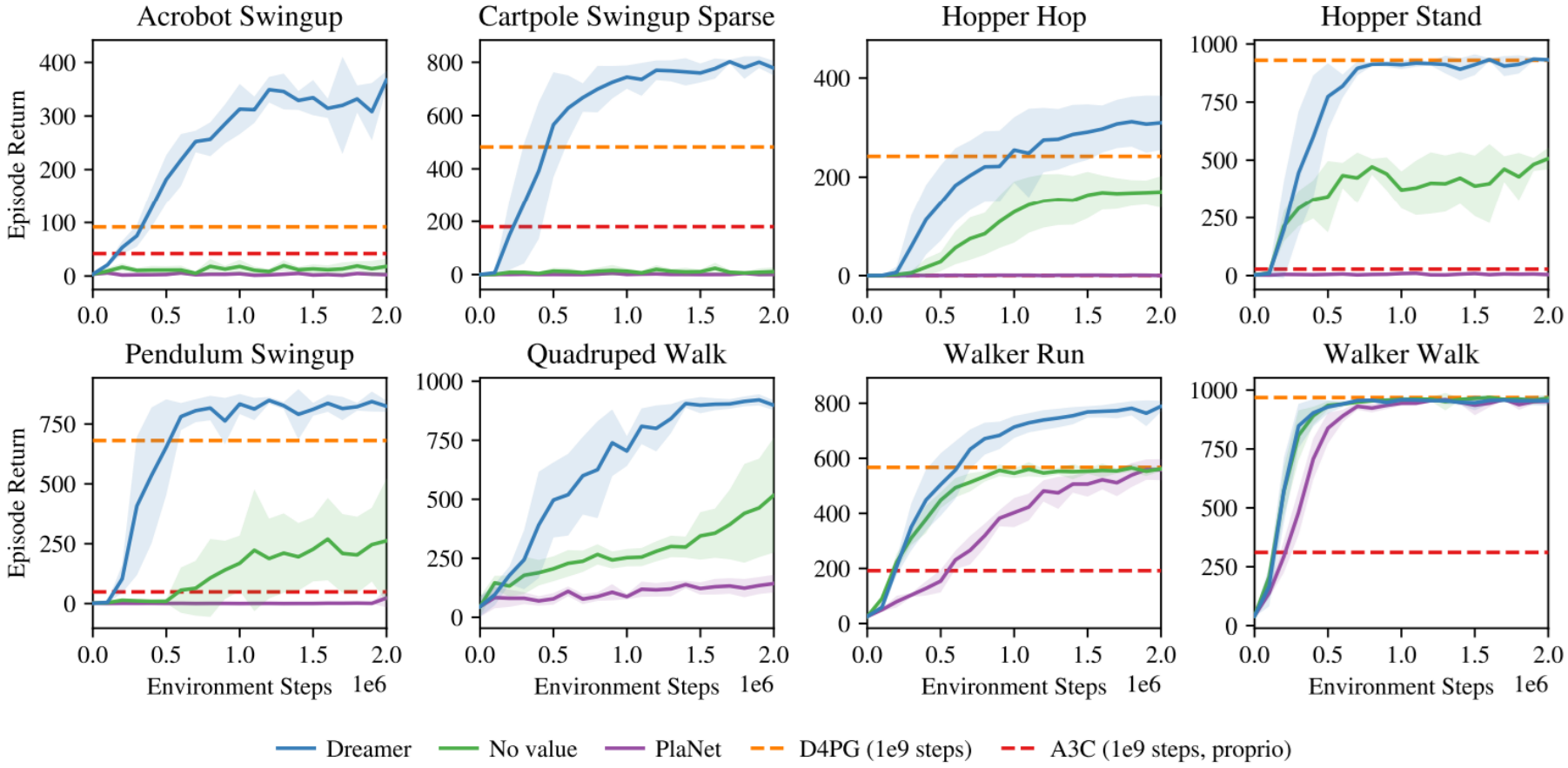
В итоге хотим:

$$\max_\phi \mathrm{E}_{q_\theta, q_\phi}\left(\sum_{\tau=t}^{t+H} \mathrm{V}_\lambda(s_\tau)\right), \; \min_\psi \mathrm{E}_{q_\theta, q_\phi}\left(\sum_{\tau=t}^{t+H} \frac{1}{2}\left\| v_\psi(s_\tau) - \mathrm{V}_\lambda(s_\tau)) \right\|^2\right)$$

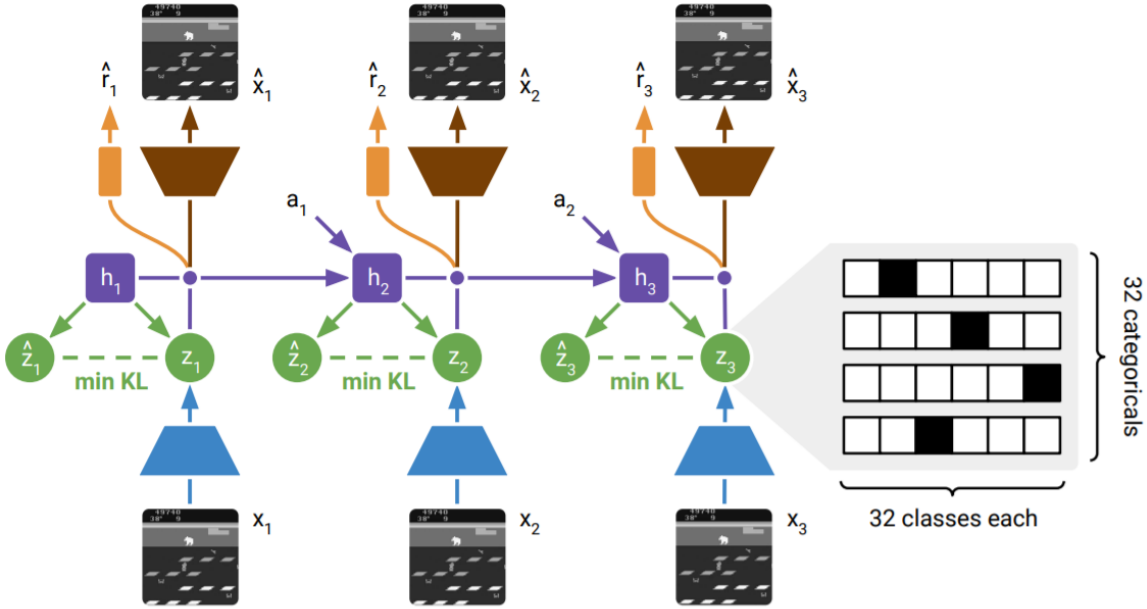
Dreamer


Dreamer (5e6 steps)
PlaNet (5e6 steps)
D4PG (1e8 steps)
A3C (1e8 steps, proprio)
Episode Return
1000
800
600
400
200
0
n/a
n/a
n/a
n/a
Cartpole Balance
Walker Stand
Cup Catch
Walker Walk
Cartpole Bal. Sparse
Reacher Easy
Quadruped Walk
Hopper Stand
Cheetah Run
Finger Turn Hard
Quadruped Run
Cartpole Swingup
Pendulum Swingup
Finger Turn Easy
Walker Run
Reacher Hard
Cartpole Swi. Sparse
Finger Spin
Hopper Hop
Acrobot Swingup

Dreamer


Acrobot Swingup
Cartpole Swingup Sparse
Hopper Hop
Hopper Stand
Pendulum Swingup
Quadruped Walk
Walker Run
Walker Walk
Episode Return
Environment Steps
1e6
Dreamer
No value
PlaNet
D4PG (1e9 steps)
A3C (1e9 steps, proprio)

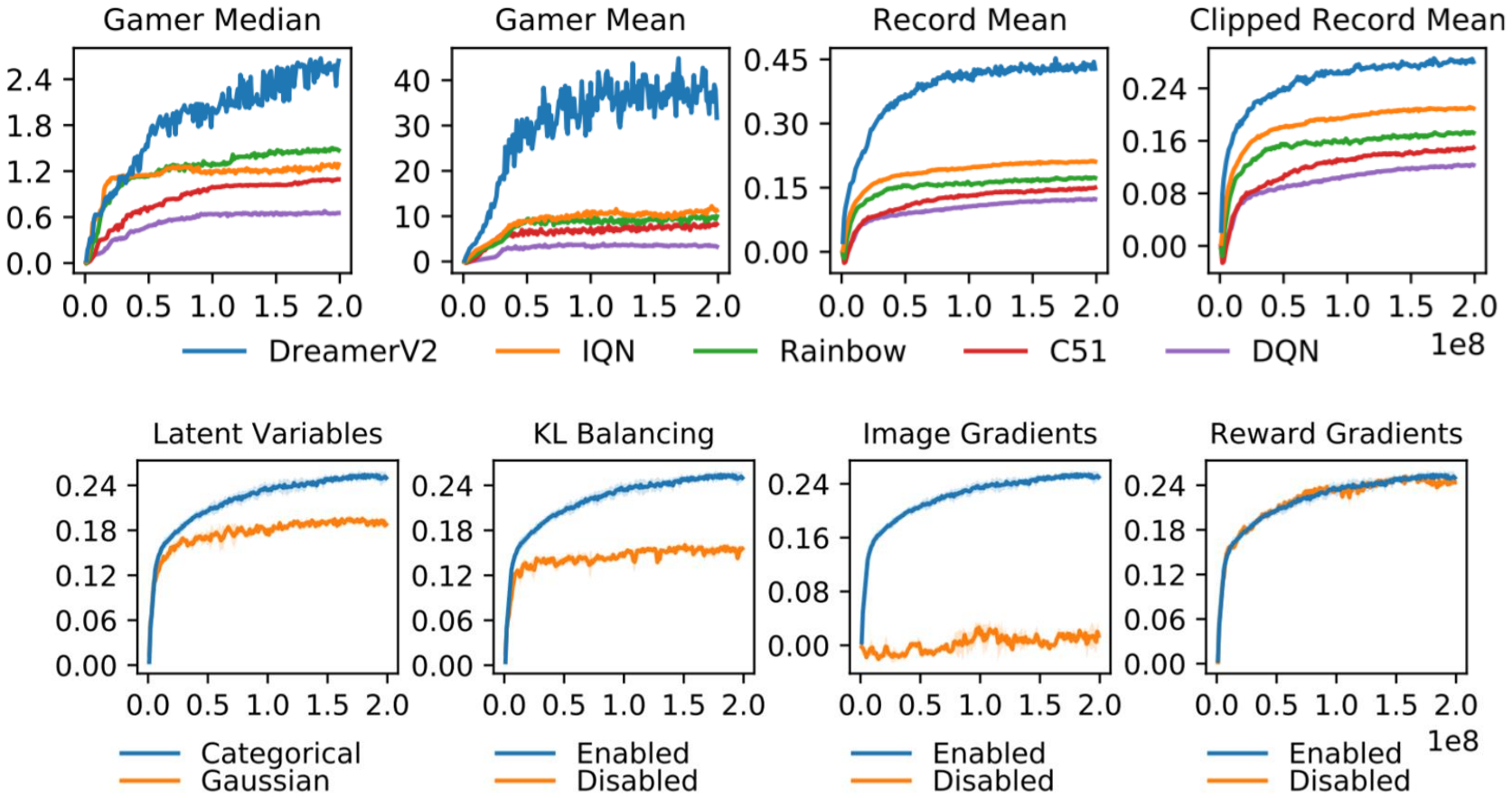
Dreamer v2



Algorithm 2: KL Balancing with Automatic Differentiation

```
kl_loss =       alpha  * compute_kl(stop_grad(approx_posterior), prior)
        + (1 - alpha)  * compute_kl(approx_posterior, stop_grad(prior))
```

Dreamer v2

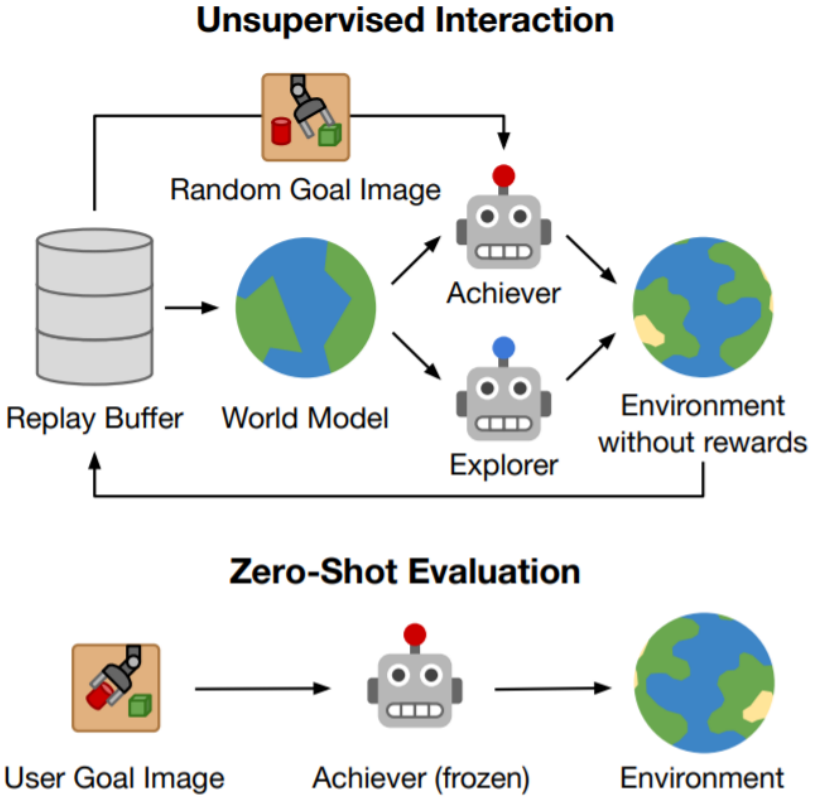

Gamer Median
Gamer Mean
Record Mean
Clipped Record Mean
DreamerV2
IQN
Rainbow
C51
DQN
1e8
Latent Variables
KL Balancing
Image Gradients
Reward Gradients
Categorical
Gaussian
Enabled
Disabled
Enabled
Disabled
Enabled
Disabled
1e8

LEXA: Model-based RL for goal reaching

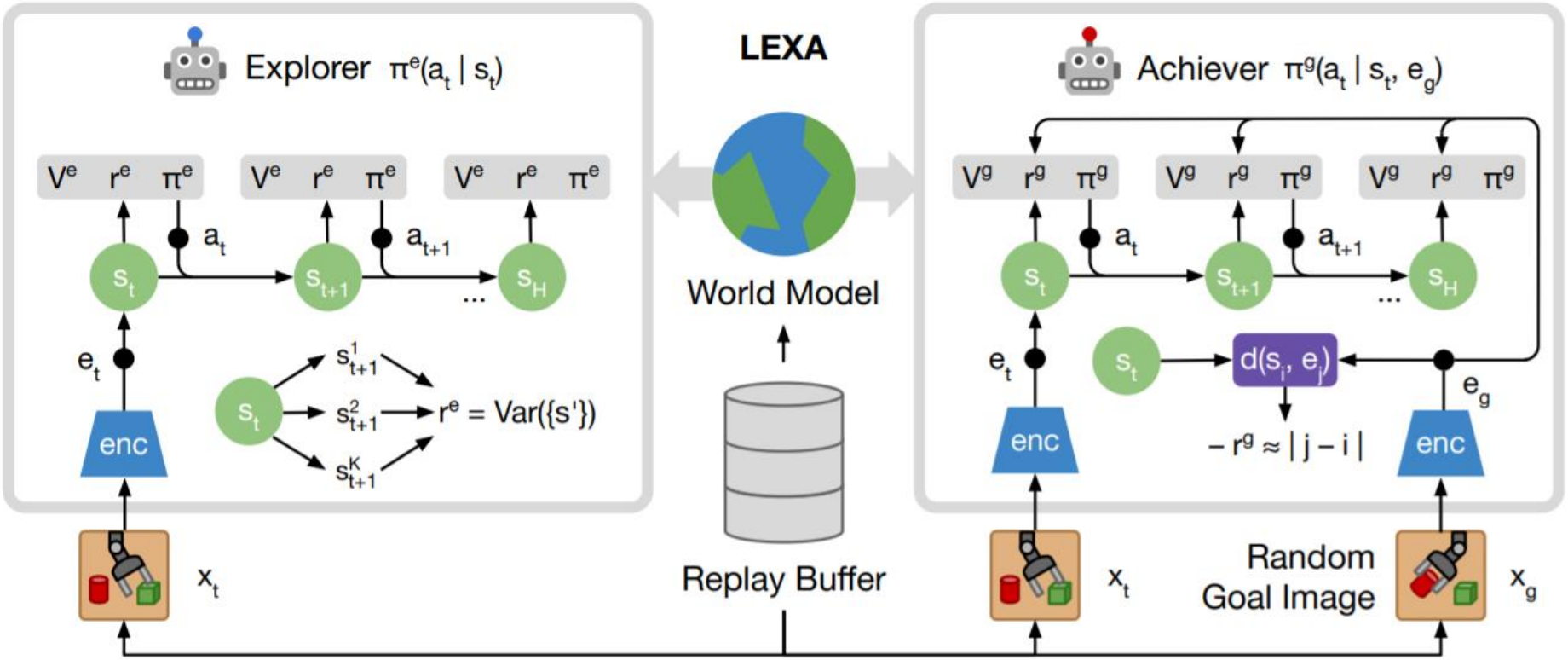
Идея:
Использовать желаемое наблюдение как цель и научить агента его достигать

Проблема 1:
Как исследовать среду, если нет набора заранее заданных целей?

Проблема 2:
Как задать функцию награды для обучения агента?



LEXA: Model-based RL for goal reaching



LEXA: Model-based RL for goal reaching

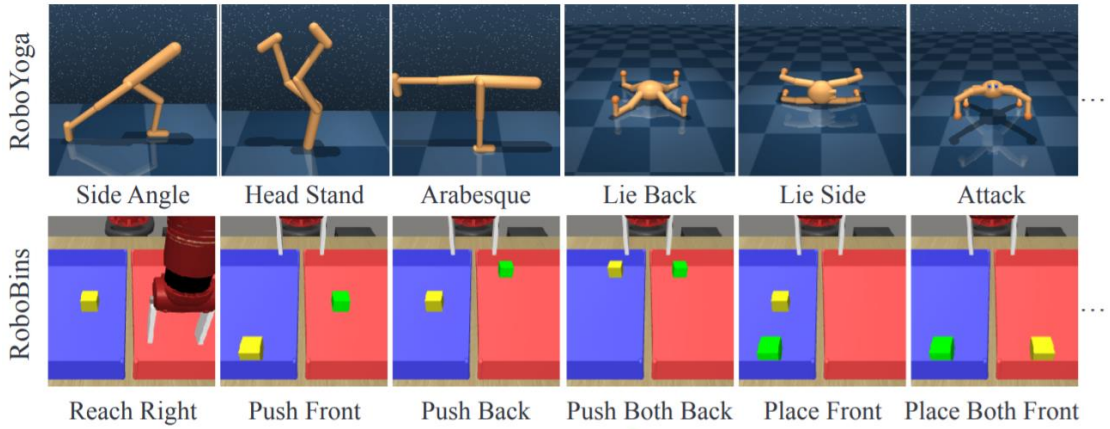
Explorer

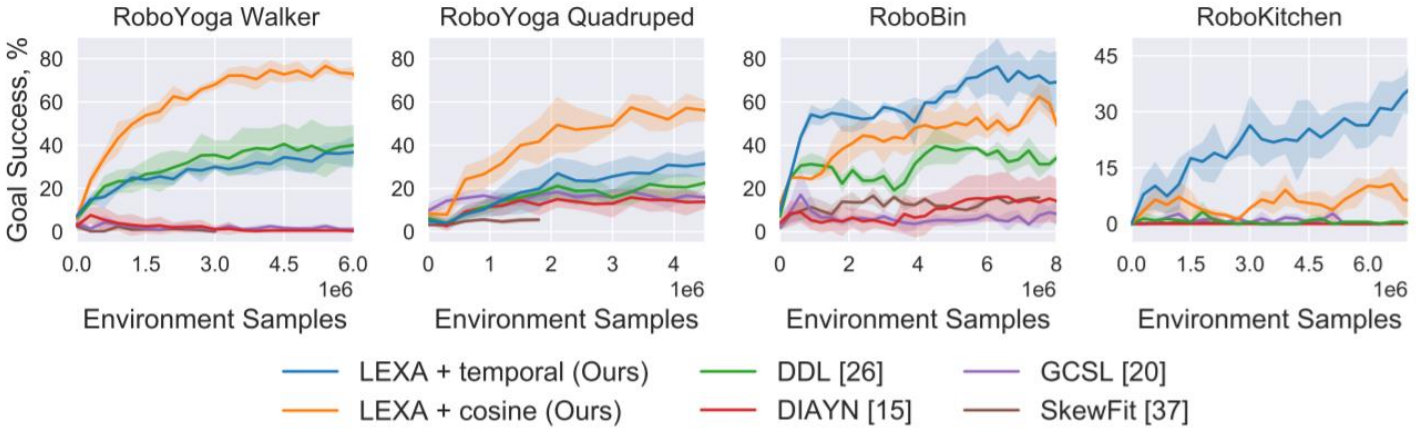
- Пытается найти состояния, которые повышают information gain
- Для этого вместе с моделью мира обучает ансамбль моделей, предсказывающих стохастическую часть состояния на следующем шаге: $p(z_{t+1} \mid s_t)$
- Variance предсказаний ансамбля используется в качестве функции награды для обучения агента, исследующего мир

Achiever

- Пытается минимизировать расстояние между целью и состоянием в латентном пространстве
- Расстояние в латентном пространстве может быть задано как cosine distance или как temporal distance
- В случае temporal distance необходимо обучить нейронную сеть, которая будет предсказывать среднее количество шагов между состояниями

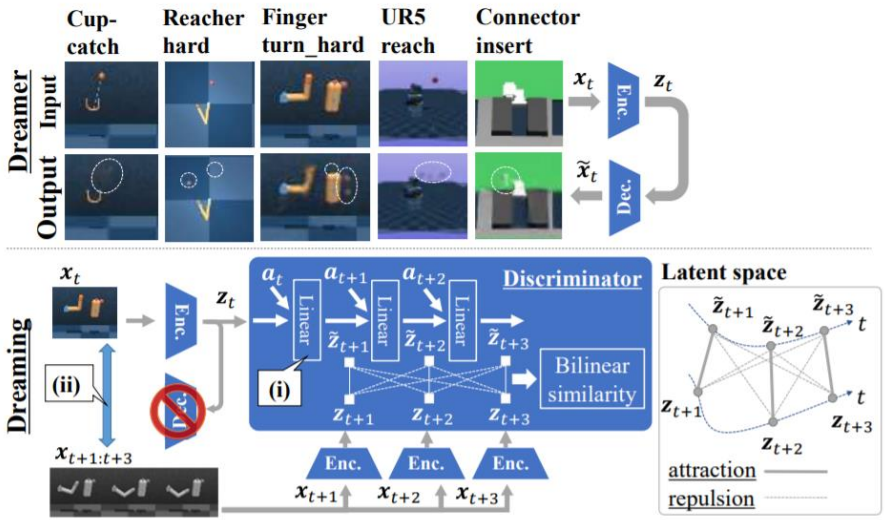
LEXA: Model-based RL for goal reaching



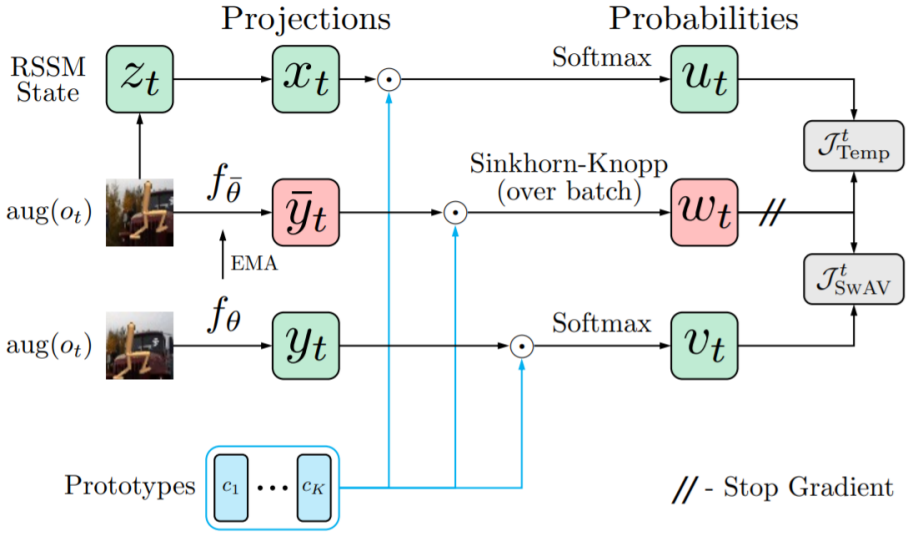


Model-based RL without reconstruction

With contrastive learning



Without contrastive learning



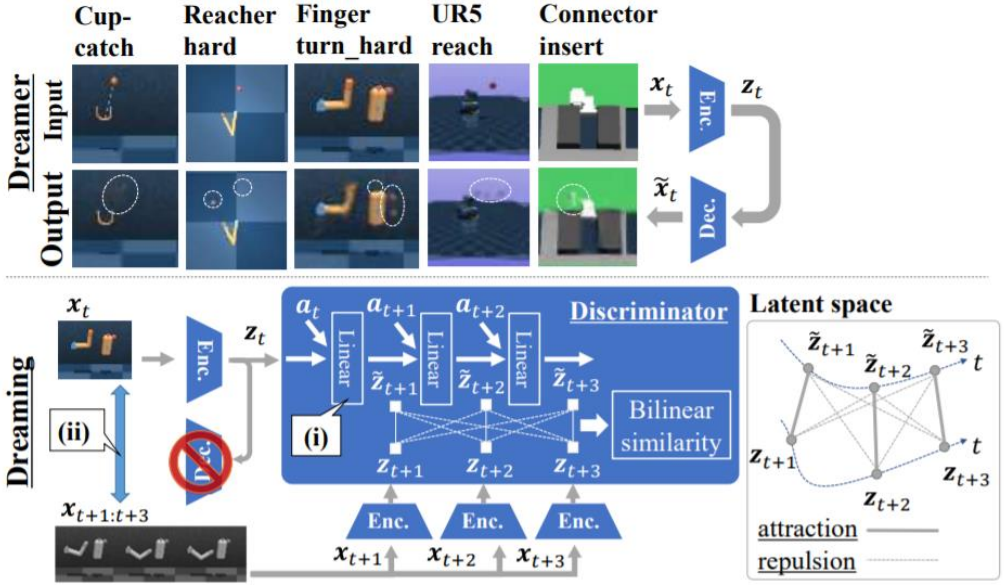
Model-based RL with contrastive learning

Идея: используем contrastive learning для получения представления состояния

В оригинальной статье уже пытались, но ничего не получилось

Чтобы это исправить, используем overshooting:

1. Закодируем последовательность переходов в среде
1. Сгенерируем последовательность переходов из того же начального состояния
1. Будем сдвигать соответствующие эмбеддинги друг к другу

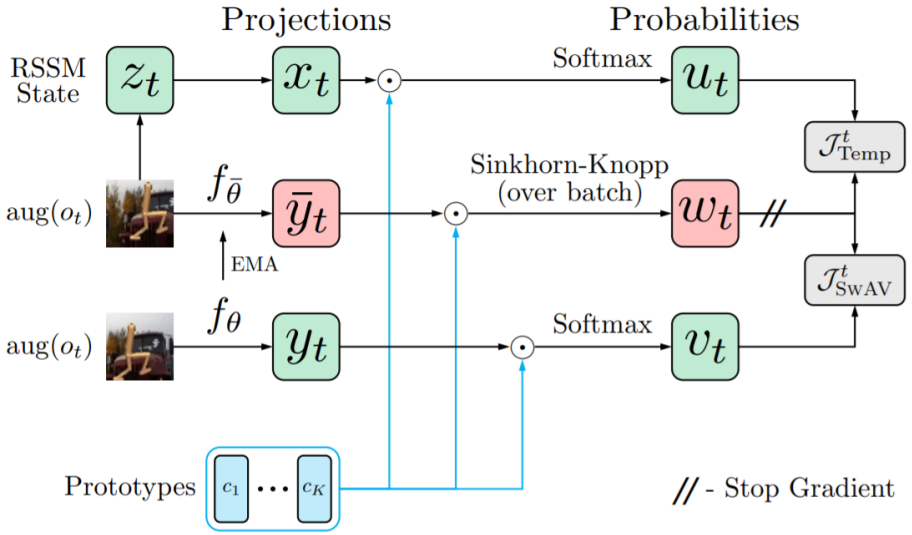


Model-based RL without contrastive learning

Идея: используем алгоритм SwAV, который строит представления состояний на основе прототипов

Отличие в том, что теперь мы будем решать задачу классификации

1. Обучаем encoder f, который учит представления для каждого изображения
1. Используем momentum encoder, который является геометрическим средним encoder'a, для определения меток класса
1. State representaton обучается таким образом, чтобы после линейной проекции решать задачу классификации

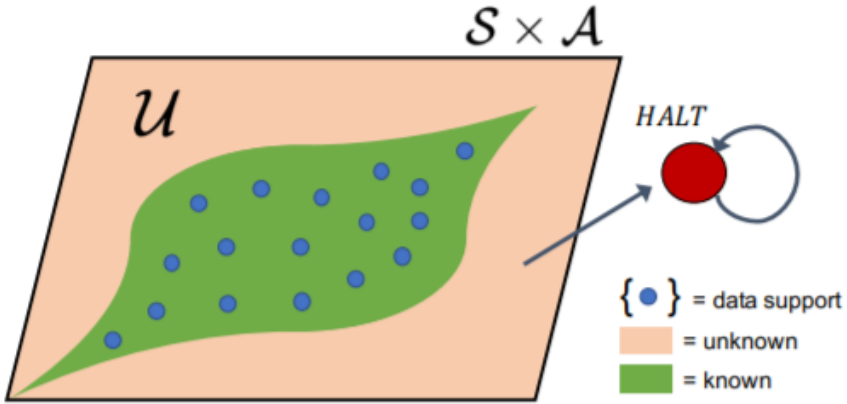


Model-Based Offline Reinforcement Learning

Идея: в случае, если модель выходит за пределы известной динамики среды, будем переводить модель в особое состояние (HALT) и давать минимальную возможную награду

Алгоритм:

1. Построить модель, приближающую динамику среды
1. Построить множество USAD, которое определяет, находится ли состояние в пределах известной динамики среды
1. Построить пессимистичный MDP
2. Использовать MDP для оптимизации политики или планирования



Model-Based Offline Reinforcement Learning

Определим USAD:

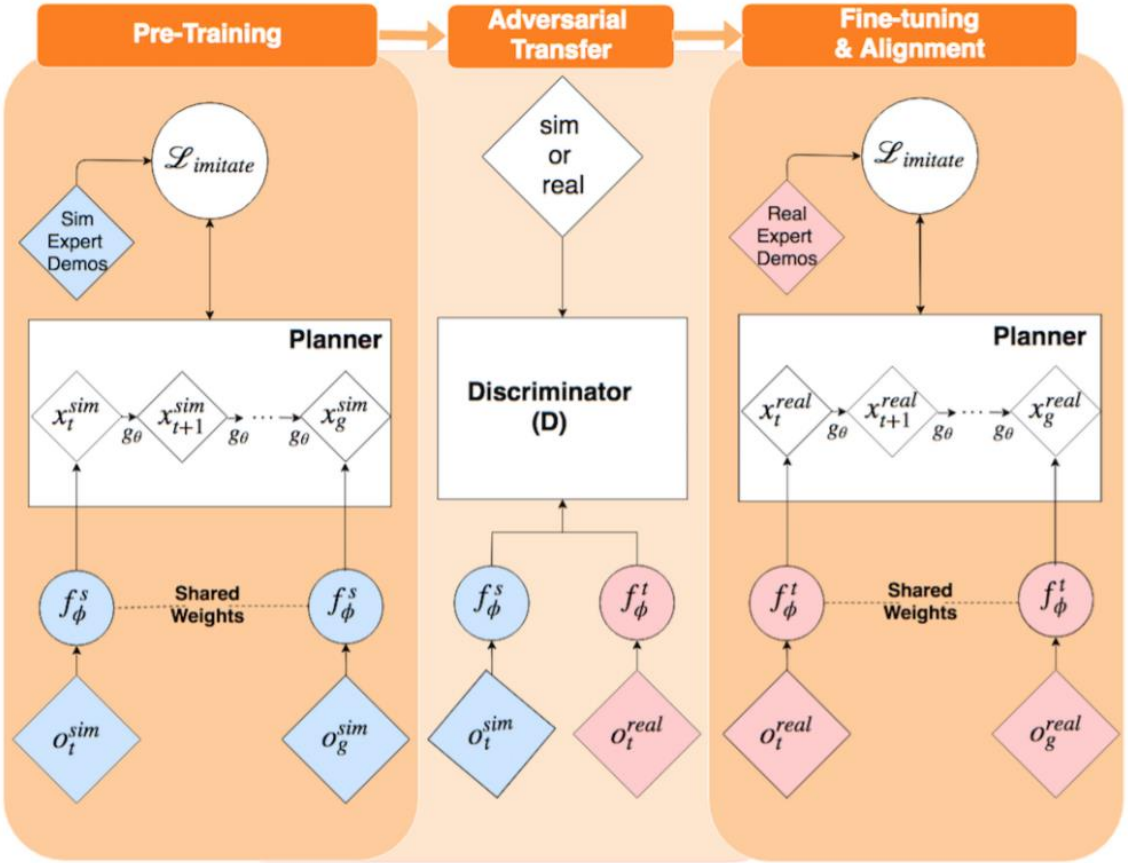
$$U^{\alpha}(s,a) = \begin{cases} \textit{FALSE (i.e. Known)} & \text{if } D_{TV}\left(\hat{P}(\cdot|s,a), P(\cdot|s,a)\right) \leq \alpha \textit{ can be guaranteed} \\ \textit{TRUE (i.e. Unknown)} & \textit{otherwise} \end{cases}$$

Однако, поскольку на практике мы не знаем распределения P(.|s, a), вместо этого будем использовать оценку неопределенности с помощью ансамбля моделей:

$$U_{\text{practical}}(s,a) = \begin{cases} \text{FALSE (i.e. Known)} & \text{if } \text{disc}(s,a) \leq \text{threshold} \\ \text{TRUE (i.e. Unknown)} & \text{if } \text{disc}(s,a) > \text{threshold} \end{cases}$$

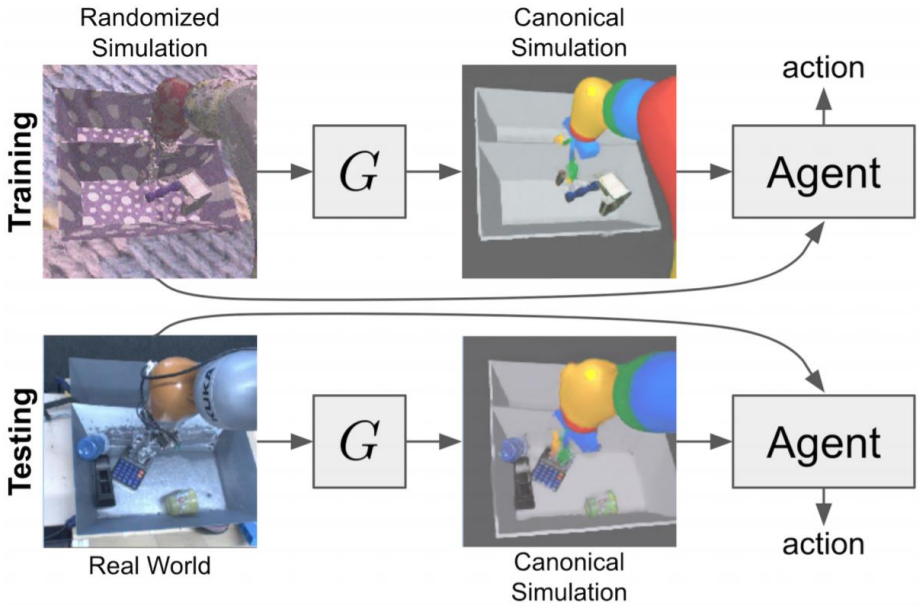
Sim-to-Real Transfer

Sim2Real via Transfer

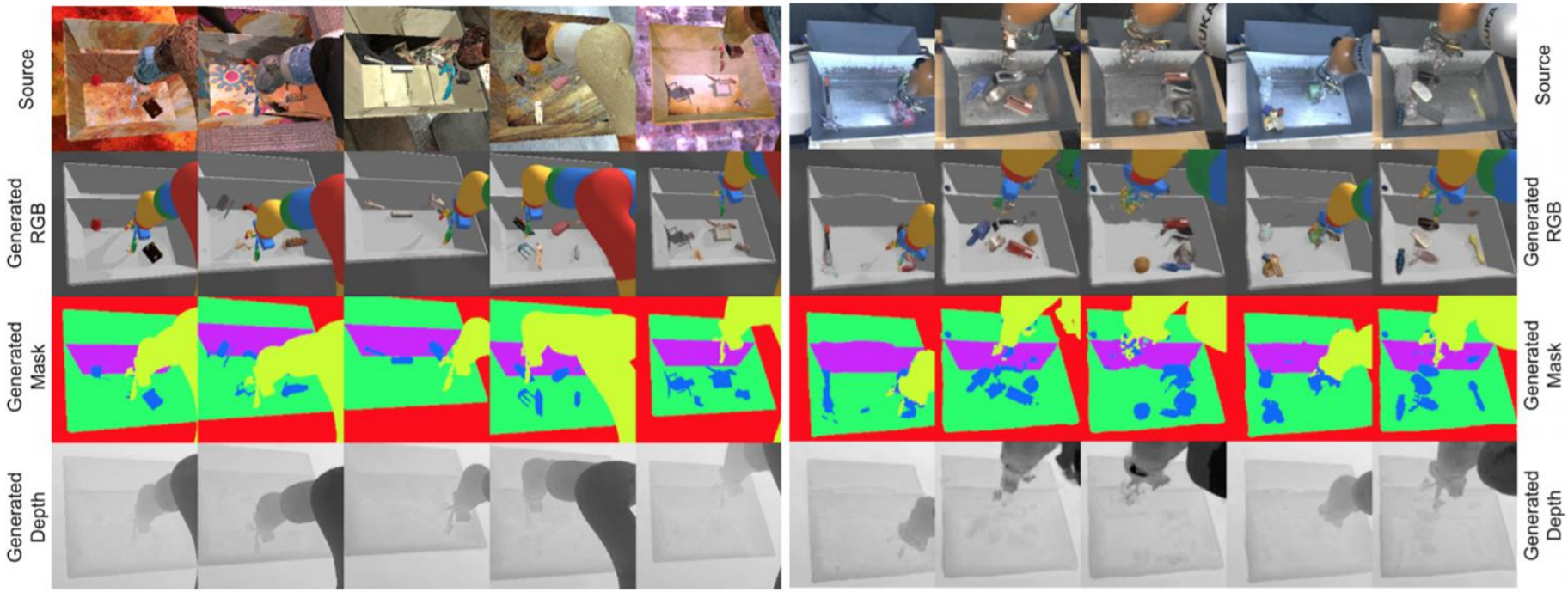


Sim2Real via Sim2Sim

Идея: рандомизировать окружение и использовать генератор, чтобы приводить картинку к стандартному виду



Sim2Real via Sim2Sim

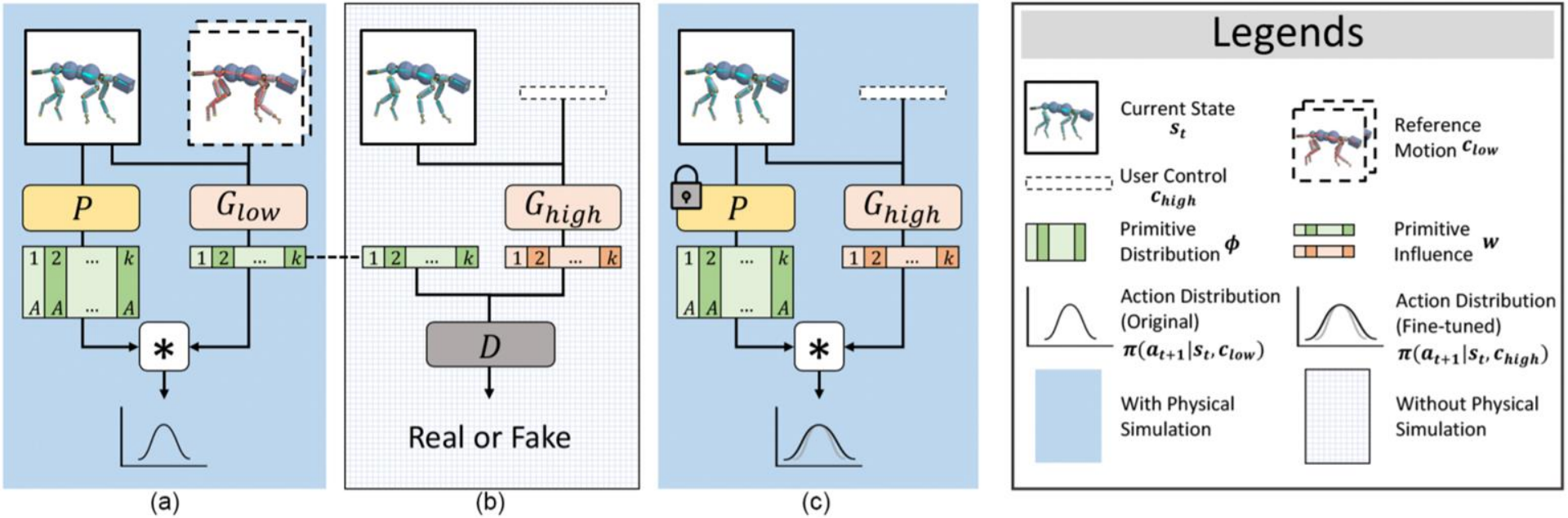


(a) Randomized-to-canonical samples.

(b) Real-to-canonical samples.

Sim2Real via Sim2Sim

Sim2Real for control adaptation



Sim2Real for control adaptation