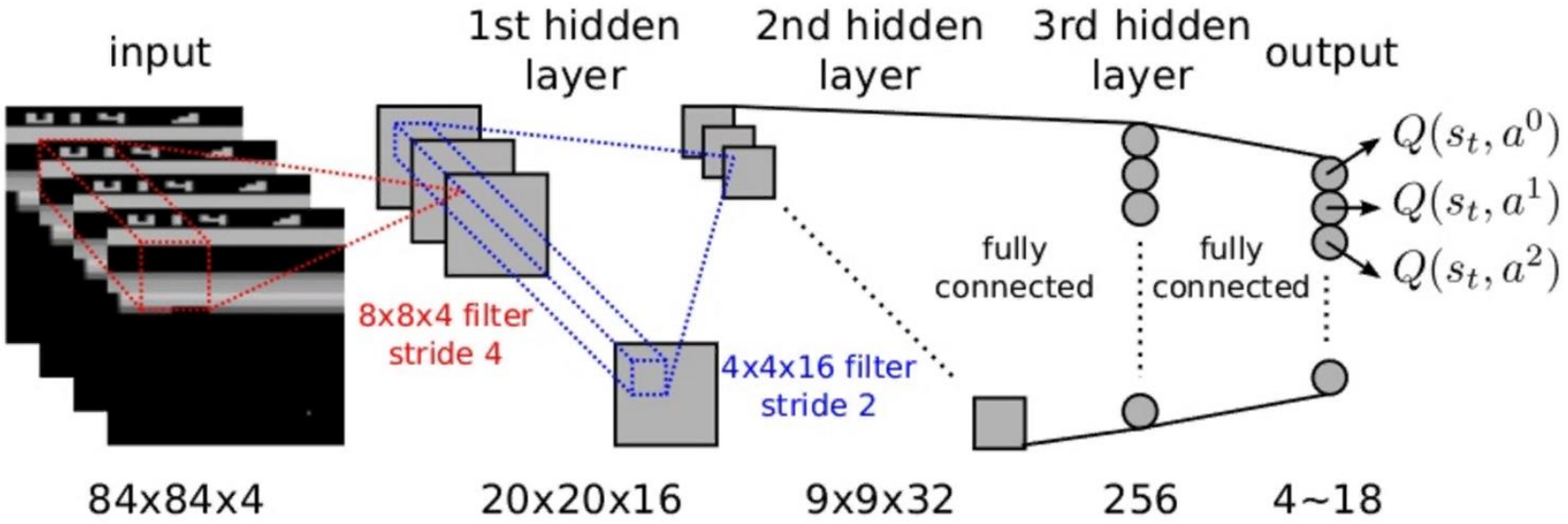
Обучение с подкреплением

Off-policy Deep RL
Learning from demonstrations

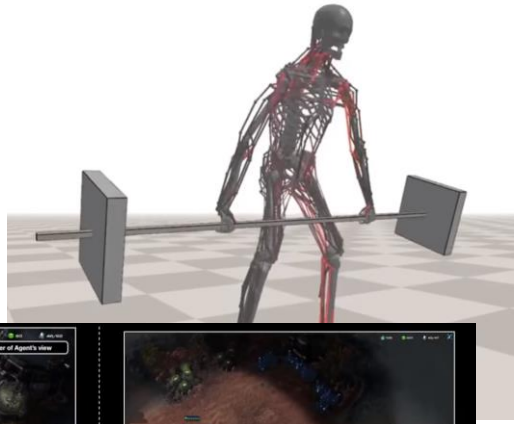
Напоминание: Deep Q-Network

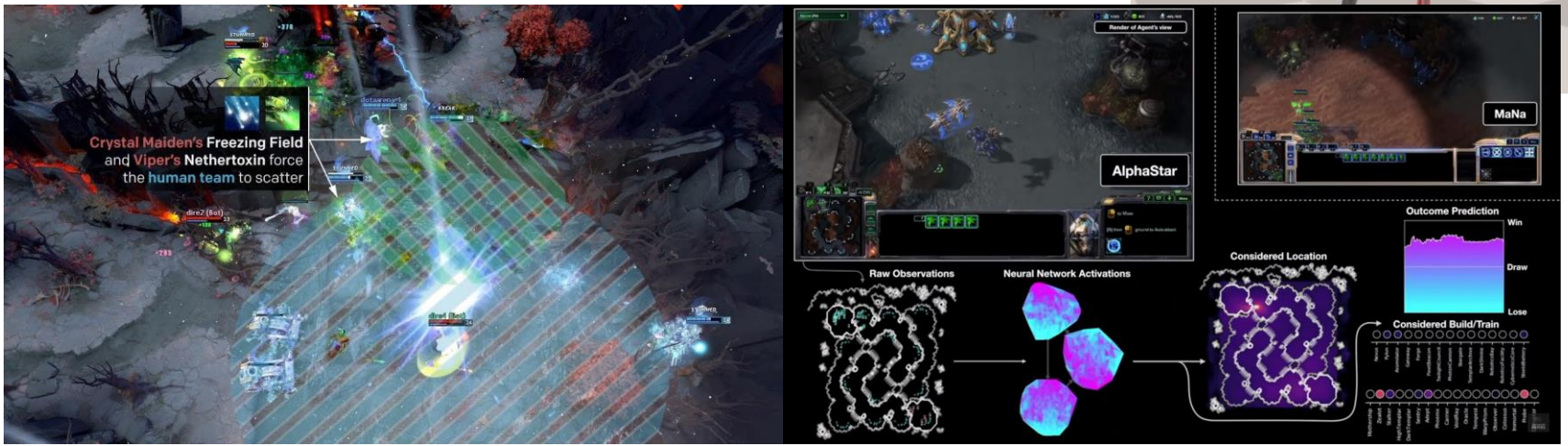
Deep Q-Network (DQN) - алгоритм, использующий для приближения Q-function нейронную сеть. Он приближает значения Q-function сразу для всех действий.



Напоминание: ограничения DQN

1) ~~Должны знать T и R~~
2) ~~Дискретное пространство состояний~~
3) Дискретное пространство действий





Напоминание: on-policy алгоритмы

Actor-Critic:

1) Использует log-derivative trick и Monte Carlo sampling для оценки градиента суммарной награды по стохастической политике
2) Может использовать только свежие данные

TRPO и NPG:

1) Жестко ограничивает KL дивергенцию между старой политикой и новой
2) Максимизирует surrogate advantage - advantage, посчитанный для старой политики
3) Использует данные, полученные с помощью предыдущей версии политики

PPO:

1) Ограничивает изменение политики с помощью clipping или penalty
2) Максимизирует surrogate advantage - advantage, посчитанный для старой политики
3) Использует данные, полученные с помощью предыдущей версии политики

Как сделать off-policy?

Deep Deterministic Policy Gradient

Алгоритм:

1. Инициализируем **Q(s, a)** и **π(s)**, создаем пустой буфер **B**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
   1. **a** := **π(s)** + **ε**, **ε** ~ N(0, **σ**)
   2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
   3. Добавляем **(s, a, s', r, d)** в **B**
   4. Сэмплим батч **b ~ B**
   5. Обновляем Q с помощью $L = \mathrm{MSE}(Q(s,a), r + \gamma \cdot Q_T(s', \pi_T(s')))$
   6. Обновляем политику, максимизируя Q
   7. **s** := **s'** если не **d**, иначе реинициализируем среду

$$Q_T = (1-\tau) \cdot Q_T + \tau \cdot Q, \;\; \pi_T = (1-\tau) \cdot \pi_T + \tau \cdot \pi$$

Deep Deterministic Policy Gradient

1) Есть целых 4 нейронных сети: Q, Q_T, $\boldsymbol{\pi}$, $\boldsymbol{\pi}_T$

1) В отличии от DQN, используется soft update весов target сетей

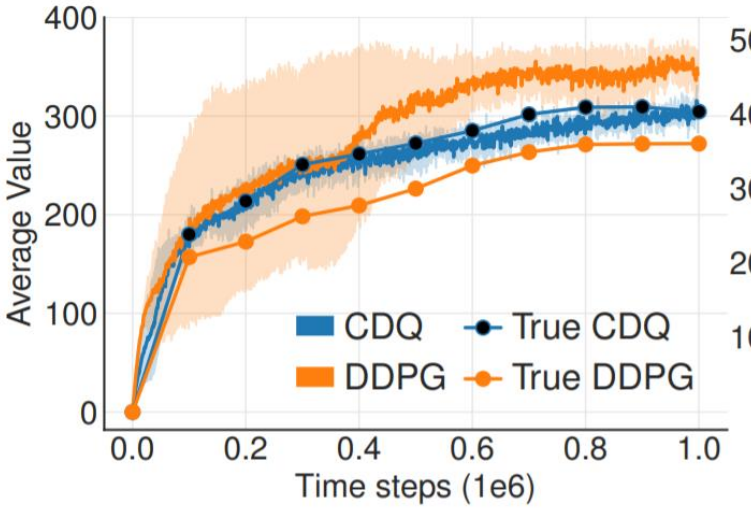
1) DDPG не является “честным” off-policy алгоритмом, т.к. не считает argmax Q-функции, но его все равно считают off-policy алгоритмом

1) Критика можно обновлять с помощью любой функции потерь, подходящей для задач регрессии

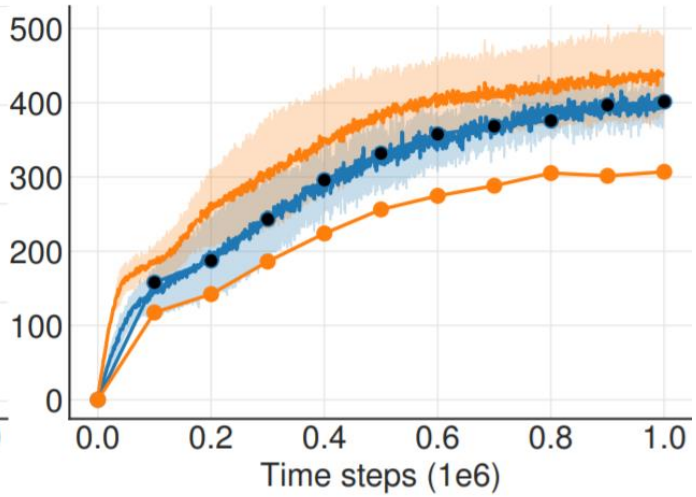
1) Использует буфер опыта. Следовательно, может использовать и любые его модификации

1) Для исследования среды использует случайный шум

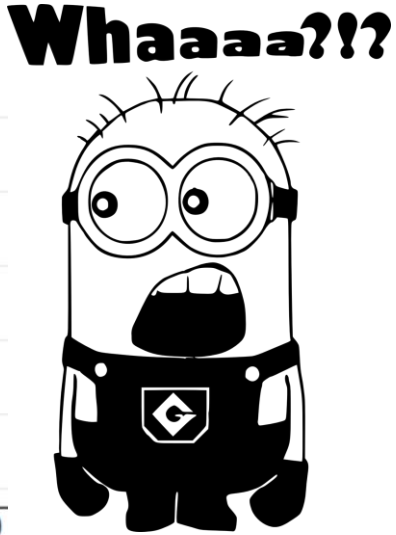
Overestimation Problem



(a) Hopper-v1



(b) Walker2d-v1



Overestimation Problem

Предположим, что мы ошибаемся на величину ε, которая распределена случайно с нулевым средним. Тогда:

$$\mathbb{E}\max_a(Q(s,a)+\varepsilon) \geq \mathbb{E}\max_a(Q(s,a))$$

Т.е., мы будем закреплять эту ошибку с каждым новым обновлением приближения Q-function.

Однако DDPG не берет максимум явно, а использует актора

Overestimation Problem

Поскольку политика стремится максимизировать приближение Q-function, то после достаточного времени обучения:

$$\mathbb{E}\hat{Q}(s, \pi(s)) \geq \mathbb{E}\hat{Q}(s, \pi^*(s))$$

С другой стороны, оптимальная политика максимизирует реальную Q-function:

$$\mathbb{E}Q(s, \pi(s)) \leq \mathbb{E}Q(s, \pi^*(s))$$

Если выполняется условие:

$$\mathbb{E}\hat{Q}(s, \pi^*(s)) \geq \mathbb{E}Q(s, \pi^*(s))$$

То можно записать такое неравенство:

$$\mathbb{E}\hat{Q}(s, \pi(s)) \geq \mathbb{E}Q(s, \pi(s))$$

Twin Delayed Deep Deterministic Policy Gradient

Алгоритм:

1. Инициализируем $\mathbf{Q_1(s, a)}$, $\mathbf{Q_2(s, a)}$ и $\boldsymbol{\pi}$**(s)**, создаем пустой буфер **B**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
    1. **a** := $\boldsymbol{\pi}$**(s)** + $\boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon}$ ~ N(0, $\boldsymbol{\sigma}$)
    2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
    3. Добавляем **(s, a, s', r, d)** в **B**
    4. Сэмплим батч **b ~ B**
    5. **a'** := $\boldsymbol{\pi}$**(s')** + clip($\boldsymbol{\varepsilon}$**', -c, c**), $\boldsymbol{\varepsilon}$**'** ~ N(0, $\boldsymbol{\sigma}$')
    6. Обновляем Q_1 и Q_2 с помощью $L = \mathrm{MSE}(Q_i(s,a), r + \gamma \min_j Q_j^T(s', a')))$
    7. Обновляем политику, максимизируя Q_1
    8. **s** := **s'** если не **d**, иначе реинициализируем среду
    9. Применяем soft update

Twin Delayed Deep Deterministic Policy Gradient

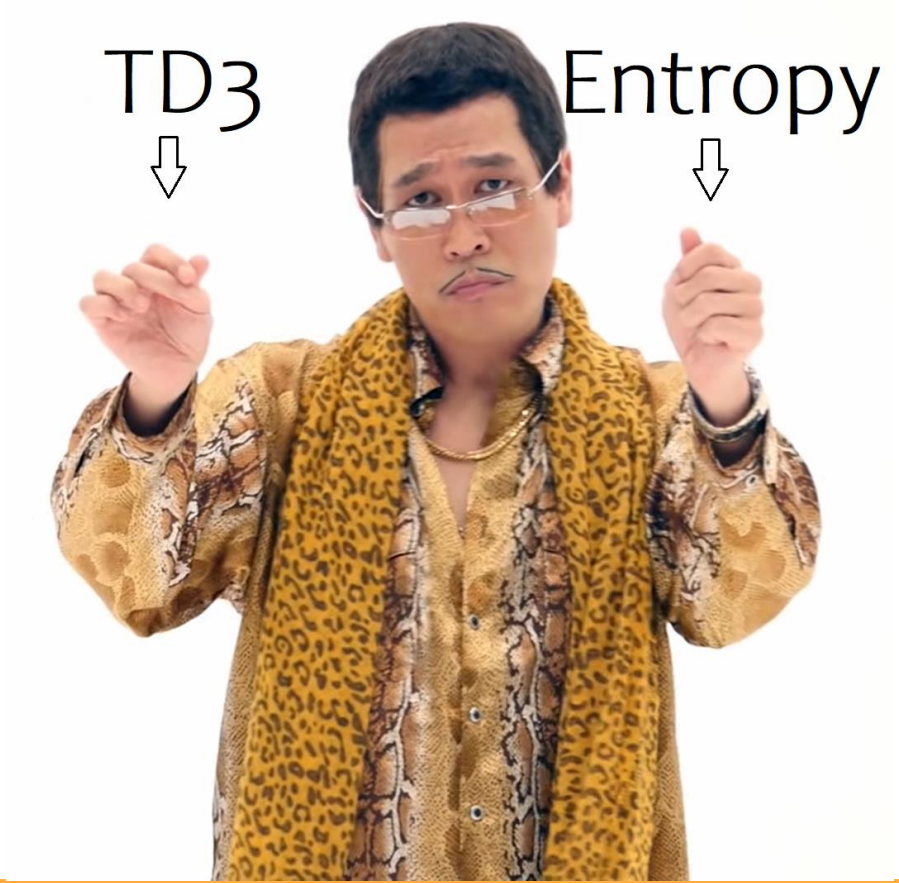
Замечания:

1) Теперь сетей 6, 4 из которых приближают Q-function, а две оставшиеся строят политику
2) Актор стремится максимизировать только первое приближения Q-function, про второе он не знает
3) У обоих критиков одинаковая loss функция

Почему это работает?

1) Q_2 будет приближением с меньшей ошибкой, т.к. мы не пытаемся его максимизировать, и неравенства со слайда 10 не работают.
2) Тем не менее, Q_2 может переоценивать Q-function. Чтобы ограничить эту переоценку, мы устанавливаем upper bound в виде значения Q_1

Off-policy: Stochastic Policy



Soft Actor-Critic

Сформулируем новую задачу оптимизации:

$$\mathbb{E}_{\tau|\pi} \sum_{i=0}^{T} \gamma^i (r_i + \alpha H(\pi(s_i))) \to \max_{\pi}$$

Можем переформулировать value function и Q-function с учетом энтропии.

Soft value function:

$$V(s) = \mathbb{E}_{\tau|\pi, s_0=s} \sum_{i=0}^{T} \gamma^i (r_i + \alpha H(\pi(s_i)))$$

Soft Q-function:

$$Q(s,a) = \mathbb{E}_{\tau|\pi, s_0=s} [r_0 + \sum_{i=1}^{T} \gamma^i (r_i + \alpha H(\pi(s_i)))]$$

Soft Actor-Critic

Soft value function → Soft Q-function

$$Q(s,a) = \mathbb{E}_{s',r}[r + \gamma \mathbb{E}_{a'}(Q(s',a') - \alpha \log \pi(a',s'))] = \mathbb{E}_{s',r}[r + \gamma V(s')]$$

Soft Q-function → Soft value function

$$V(s) = \mathbb{E}_{a\sim\pi(s)}\mathbb{E}_{s',r}[(r - \alpha \log \pi(a,s)) + \gamma V(s')] = \mathbb{E}_{a\sim\pi(s)}[Q(s,a) - \alpha \log \pi(a,s)]$$

Проблема: не хотим считать матожидание по политике в континуальном случае

Reparametrization trick:

$$V(s) = \mathbb{E}_{a\sim\pi(s)}[Q(s,a) - \alpha \log \pi(a,s)] = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}[Q(s, f_\pi(s,\epsilon)) - \alpha \log \pi(f_\pi(s,\epsilon), s)]$$

где

$$f_\pi(s,\epsilon) = \tanh(\mu_\pi(s) + \epsilon \cdot \sigma_\pi(s))$$

Soft Actor-Critic

1. Инициализируем $\mathbf{Q_1(s, a)}$, $\mathbf{Q_2(s, a)}$ и стохастическую $\boldsymbol{\pi}\mathbf{(s)}$, создаем пустой буфер **B**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
   1. **a** ~ $\boldsymbol{\pi}$**(s)**
   2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
   3. Добавляем **(s, a, s', r, d)** в **B**
   4. Сэмплим батч **b ~ B**
   5. **a'** ~ $\boldsymbol{\pi}$**(s')**
   6. Обновляем Q_1 и Q_2 с помощью функции потерь:

$$L = \mathrm{MSE}(Q_i(s,a), r - \alpha \log \pi(a'|s') + \gamma \min_j Q_j^T(s', a'))$$

   1. Обновляем политику, максимизируя Q_1 с учетом энтропии

$$\mathbb{E}_{a\sim\pi} Q_1(s,a) - \alpha \log \pi(a,s) \to \max_\pi$$

   1. **s** := **s'** если не **d**, иначе реинициализируем среду
   2. Применяем soft update

Soft Actor-Critic

Замечания:

1) За основу взят алгоритм TD3 и во многом на него похож

1) Политика снова стохастическая

1) Алгоритм считается off-policy алгоритмом, т.к. использует предыдущий опыт, хотя учится в предположении текущей политики

1) Коэффициент энтропии можно менять со временем или делать константным

Ограничения

1) ~~Должны знать T и R~~
2) ~~Дискретное пространство состояний~~
3) ~~Дискретное пространство действий~~

Что дальше?

Ограничения

1) ~~Должны знать T и R~~
2) ~~Дискретное пространство состояний~~
3) ~~Дискретное пространство действий~~

Как долго алгоритмы учатся?

Rainbow DQN: ~$2 \cdot 10^8$ шагов в Atari
SAC и PPO: ~$3 \cdot 10^6$ шагов в MuJoCo

Ограничения

1) ~~Должны знать T и R~~
2) ~~Дискретное пространство состояний~~
3) ~~Дискретное пространство действий~~

Как долго алгоритмы учатся?

Rainbow DQN: ~$2 \cdot 10^8$ шагов в Atari
SAC и PPO: ~$3 \cdot 10^6$ шагов в MuJoCo

Насколько это быстро?

Если шаги совершаем в реальном времени, и 1 шаг - это 1/20 секунды, то потребуется ~2778 часов для обучения Rainbow DQN

Ограничения

1) ~~Должны знать T и R~~
2) ~~Дискретное пространство состояний~~
3) ~~Дискретное пространство действий~~

Как долго алгоритмы учатся?

Rainbow DQN: ~$2 \cdot 10^8$ шагов в Atari
SAC и PPO: ~$3 \cdot 10^6$ шагов в MuJoCo

Насколько это быстро?

Если шаги совершаем в реальном времени, и 1 шаг - это 1/20 секунды, то потребуется ~2778 часов для обучения Rainbow DQN

Более того, для решения более сложных задач требуется еще больше опыта, а некоторые окружения решить таким подходом совсем не получится

Перерыв

Обучение по демонстрациям

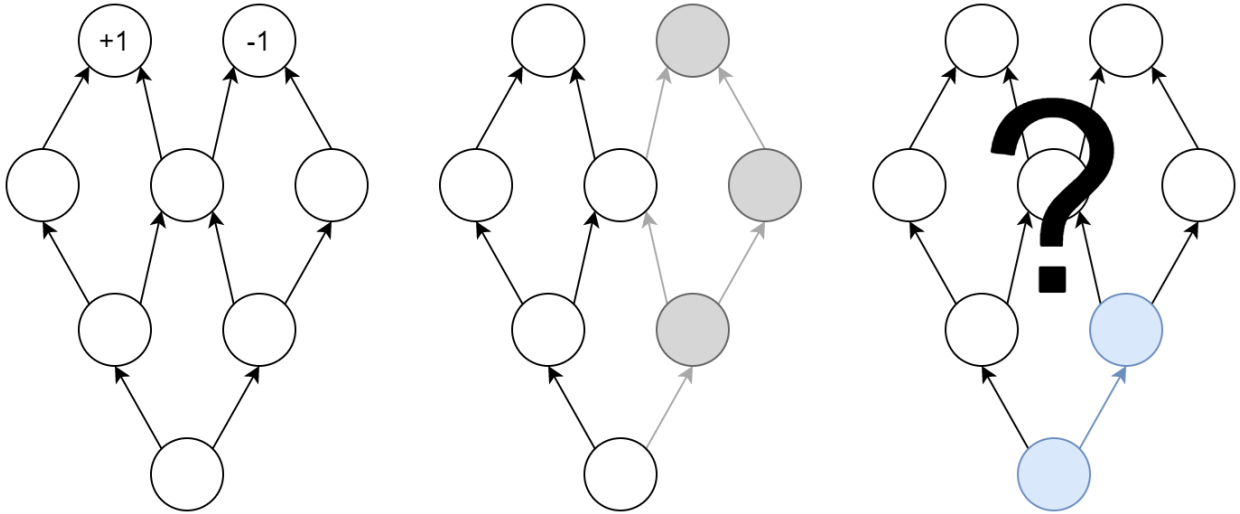
Предположим, что существует эксперт, который умеет решать задачу и может продемонстрировать несколько решений

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

Behavioral Cloning

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

Проблема:



Inverse Reinforcement Learning

Идея: научимся отличать политику эксперта от остальных политик с помощью функции похожести c(s,a)

Найдем функцию, которая была бы как можно меньше для политики эксперта и как можно больше для любой другой политики:

$$\max_c[-(\mathbb{E}_{s,a\sim\pi_{\text{expert}}}c(s,a)) + \min_\pi(-H(\pi) + \mathbb{E}_{s,a\sim\pi}c(s,a)))]$$

Минимизируя такую функцию при помощи RL, мы получим политику, которая будет похожа на эксперта.

Проблема: c(s,a) может быть бесконечной, поэтому добавим регуляризацию

$$\max_c[-\psi(c) - \mathbb{E}_{\pi_{\text{expert}}}c(s,a) + \min_\pi(-H(\pi) + \mathbb{E}_\pi c(s,a))]$$

Generative Adversarial Imitation Learning

Введем понятие occupancy measure:

$$\rho_\pi(s,a) = \pi(a|s) \sum \gamma^t p(s_t = s|\pi)$$

И предположим, что политика однозначно задается ее occupancy measure.

Кроме того, определим выпуклое сопряжение функции $f : \mathbb{R}^{S\times A} \to \bar{\mathbb{R}}$ как:

$$f^*(x) = \sup_y x^T y - f(y)$$

Тогда решением задачи будет (без доказательства):

$$\mathrm{RL}(\mathrm{IRL}_\psi(\pi_{\mathrm{expert}})) = \arg\min_\pi -H(\pi) + \psi^*(\rho_\pi - \rho_{\pi_{\mathrm{expert}}})$$

Таким образом, переходим к задаче occupancy measure matching

Generative Adversarial Imitation Learning

В итоге наша цель заключается в нахождении оптимума:

$$\arg\min_{\pi} d_{\psi}(\rho_{\pi}, \rho_{\pi_{\text{expert}}}) - H(\pi)$$

Если определим регуляризацию следующим образом:

$$\psi_{\text{GA}}(c) \triangleq \begin{cases} \mathbb{E}_{\pi_E}[g(c(s,a))] & \text{if } c < 0 \\ +\infty & \text{otherwise} \end{cases} \quad \text{where } g(x) = \begin{cases} -x - \log(1 - e^x) & \text{if } x < 0 \\ +\infty & \text{otherwise} \end{cases}$$

То можем обучать эту функцию как дискриминатор:

$$\psi^*_{\text{GA}}(\rho_{\pi} - \rho_{\pi_E}) = \max_{D \in (0,1)^{\mathcal{S} \times \mathcal{A}}} \mathbb{E}_{\pi}[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))]$$

Generative Adversarial Imitation Learning

Для оптимизации политики будем использовать определенную ранее регуляризацию:

$$\arg\min_{\pi} d_{\psi}(\rho_{\pi}, \rho_{\pi_{\text{expert}}}) - \lambda H(\pi) = \arg\min_{\pi} \psi^{*}_{\text{GA}}(\rho_{\pi}, \rho_{\pi_{\text{expert}}}) - \lambda H(\pi)$$

Поскольку мы уже знаем, что выпуклая сопряженная функция для регуляризации - это дискриминатор, то:

$$\arg\min_{\pi} \psi^{*}_{\text{GA}}(\rho_{\pi}, \rho_{\pi_{\text{expert}}}) - \lambda H(\pi) = \arg\min_{\pi} D_{\text{JS}}(\rho_{\pi}, \rho_{\pi_{\text{expert}}}) - \lambda H(\pi)$$

Generative Adversarial Imitation Learning

Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))]$$

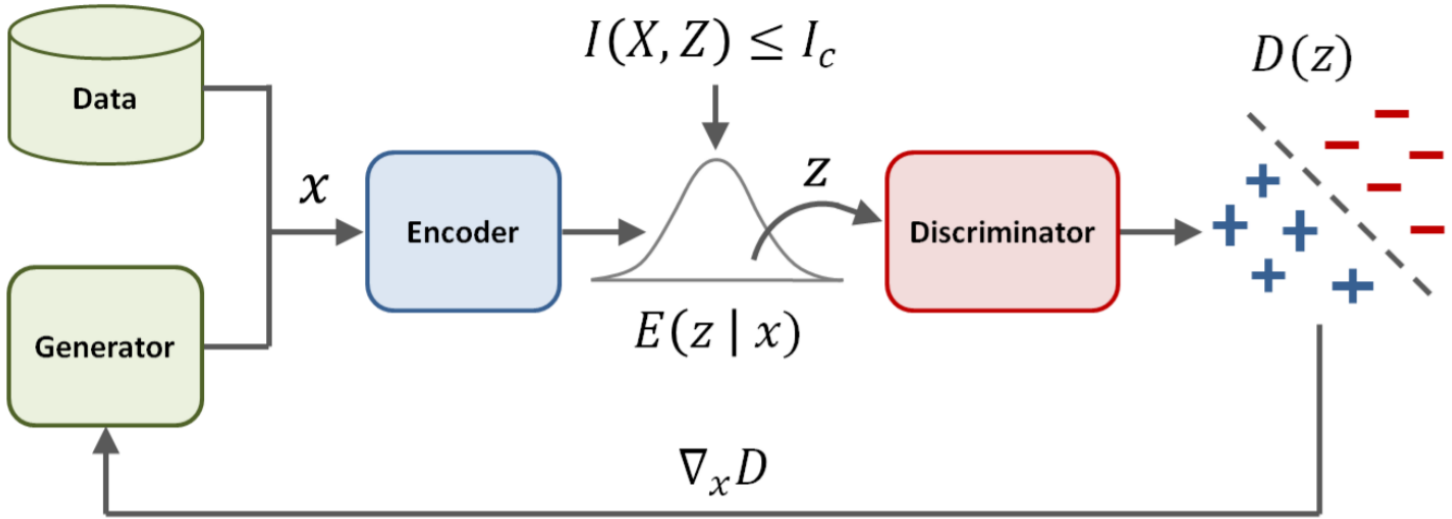
5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta),$$
$$\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}]$$

6: **end for**

VAIL

Проблема: дискриминатор может переобучиться



Идея: использовать information bottleneck

VAIL

Хотим решать задачу в таком виде:

$$\tilde{J}(q, E) = \min_{q,E} \quad \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p(\mathbf{x},\mathbf{y})} \left[\mathbb{E}_{\mathbf{z}\sim E(\mathbf{z}|\mathbf{x})} \left[-\log q(\mathbf{y}|\mathbf{z})\right]\right]$$
$$\text{s.t.} \quad \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \left[\mathrm{KL}\left[E(\mathbf{z}|\mathbf{x})||r(\mathbf{z})\right]\right] \leq I_c.$$

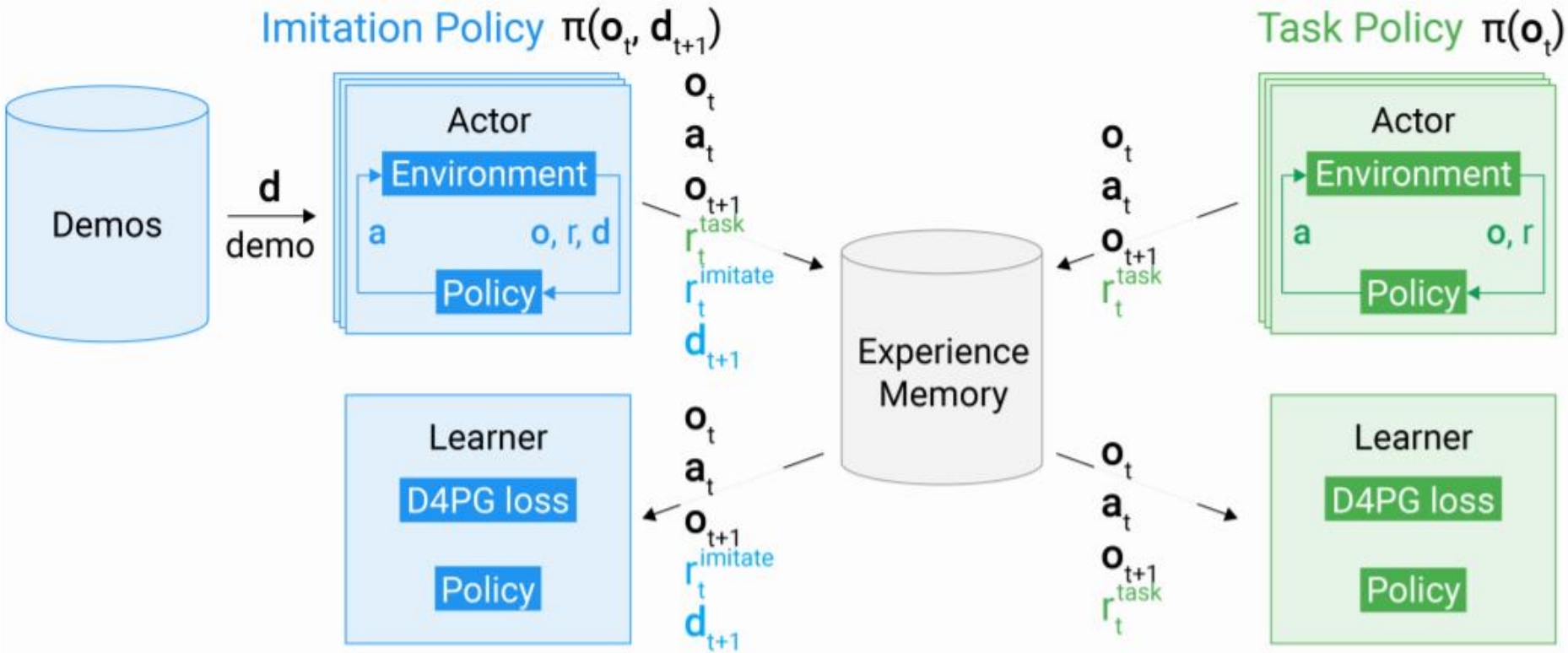
Решать задачу условной оптимизации сложно, поэтому используем регуляризацию с адаптивным коэффициентом:

$$\min_{q,E} \quad \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p(\mathbf{x},\mathbf{y})} \left[\mathbb{E}_{\mathbf{z}\sim E(\mathbf{z}|\mathbf{x})} \left[-\log q(\mathbf{y}|\mathbf{z})\right]\right] + \beta \left(\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \left[\mathrm{KL}\left[E(\mathbf{z}|\mathbf{x})||r(\mathbf{z})\right]\right] - I_c\right)$$

Где коэффициент регуляризации обновляется следующим образом:

$$\beta \leftarrow \max\left(0,\ \beta + \alpha_\beta \left(\mathbb{E}_{\mathbf{x}\sim \tilde{p}(\mathbf{x})} \left[\mathrm{KL}\left[E(\mathbf{z}|\mathbf{x})||r(\mathbf{z})\right]\right] - I_c\right)\right)$$

MetaMimic


Imitation Policy $\pi(o_t, d_{t+1})$
Task Policy $\pi(o_t)$
Demos
d
demo
Actor
Environment
a
o, r, d
Policy
Actor
Environment
a
o, r
Policy
o_t
a_t
o_{t+1}
r_t^{task}
$r_t^{imitate}$
d_{t+1}
Experience Memory
Learner
D4PG loss
Policy
Learner
D4PG loss
Policy

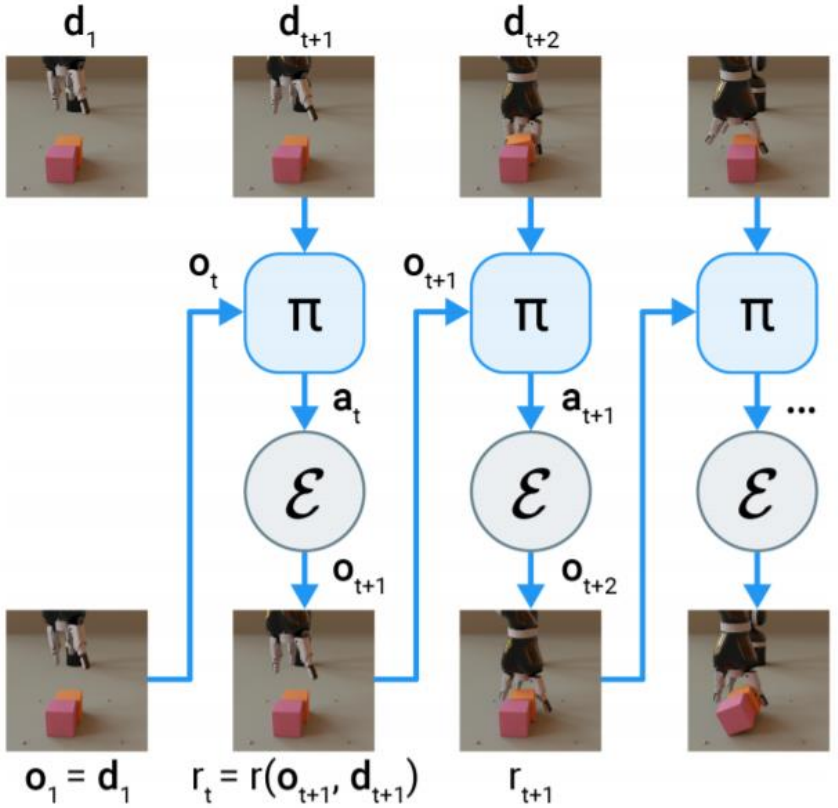
MetaMimic

Пытаемся решить такую задачу:

$$\arg\max_{\pi} \mathbb{E}_{d\sim D,\tau\sim\pi(.,d)} \sum \gamma^t \mathrm{sim}(d_t, o_t)$$

Sim - это некоторая метрика похожести демонстрации и наблюдения

При этом итоговый агент (в идеале) будет способен повторять любую демонстрацию



DeepMimic

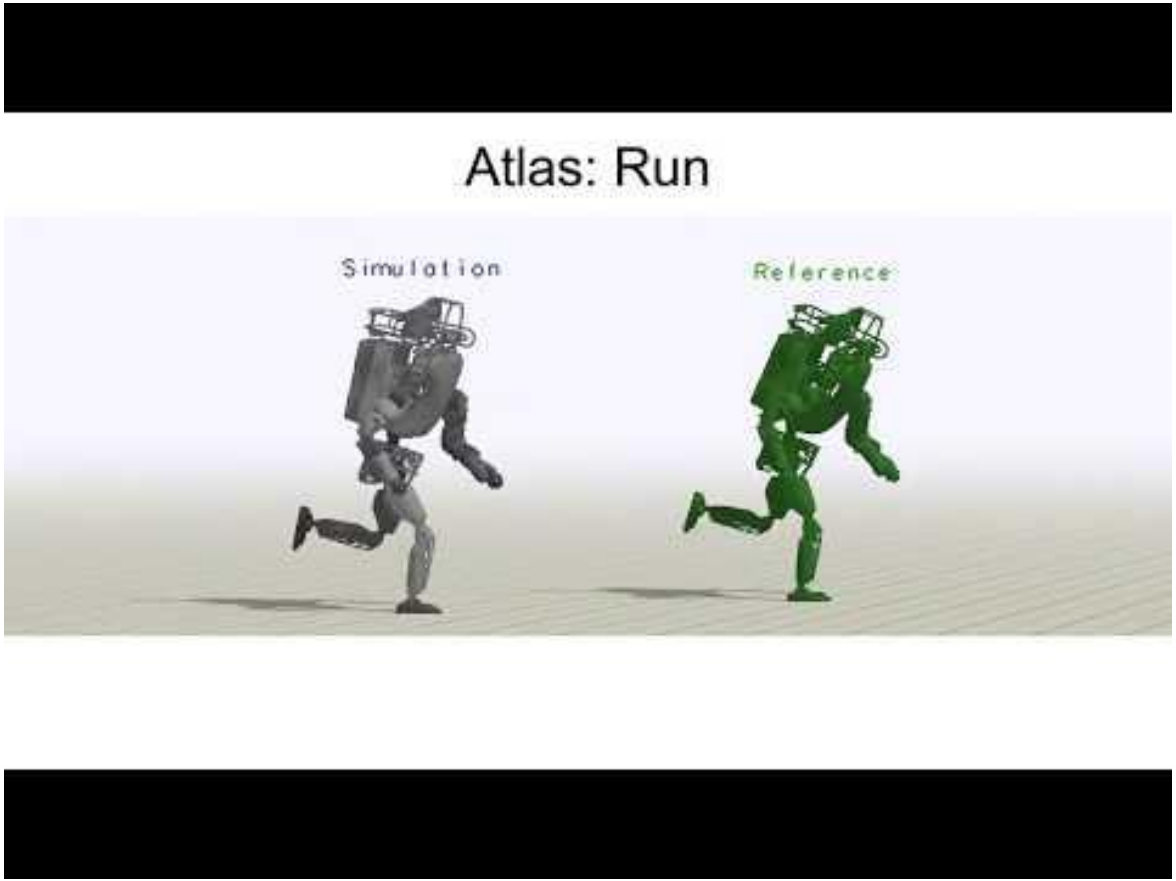
Идея в том, чтобы давать агенту награду за следование демонстрации и за выполнение поставленной задачи:

$$r_t = w_t^I \cdot r_t^I + w_t^G \cdot r_t^G$$

Награда за имитацию задается вручную и тем больше, чем лучше агент повторяет положение тела в демонстрации в момент времени t. Также можно использовать несколько демонстраций, для этого можно выбрать максимум награды по всем демонстрациям в момент t.

Награда за достижение цели соответствует выполнению задач среды.

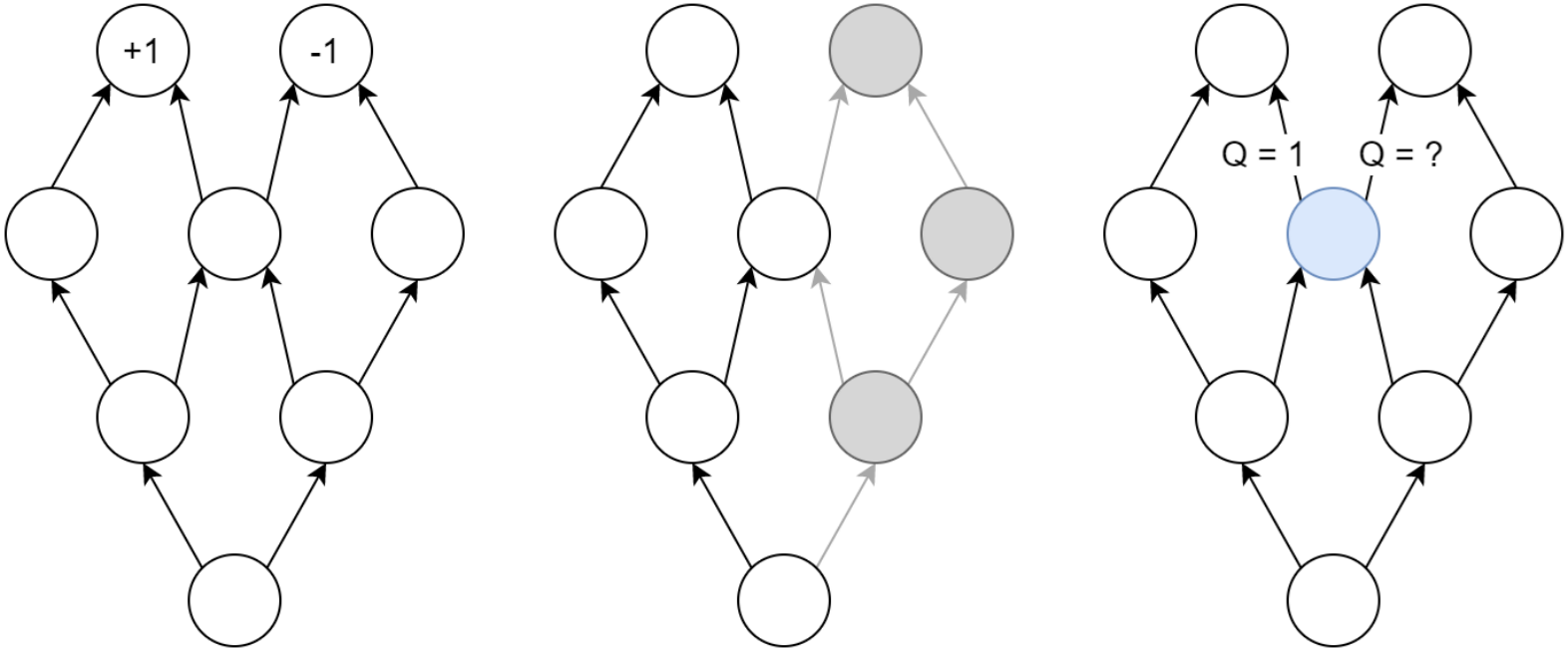
DeepMimic



Deep Q-learning from Demonstrations

Идея: будем обучать DQN по данным из демонстраций эксперта

Проблема:



Deep Q-learning from Demonstrations

Идея: будем обучать DQN по данным из демонстраций эксперта

Проблема:
Опыт эксперта покрывает не все возможные пары (s, a) в окружении

Решение:

1. Добавим дополнительную составляющую в функцию потерь:

$$L_{\mathrm{E}} = \max_{a}[Q(s,a) + l(a, a_{\mathrm{E}})] - Q(s, a_{\mathrm{E}})$$

Margin function — 0 если действия равны, > 0 иначе

Действие эксперта

1. Добавим L2 регуляризацию, чтобы избежать переобучения
2. Добавим n-step return чтобы уменьшить overestimation error

Deep Q-learning from Demonstrations

Алгоритм:

1. Добавить в буфер опыта опыт эксперта
2. В течении T1 итераций:
    1. Получить батч из буфера опыта с приоритетами
    2. Посчитать функцию потерь:
    $$\mathrm{MSE}(Q(s,a), R_1(s)) + \lambda_1 \cdot \mathrm{MSE}(Q(s,a), R_n(s)) + \lambda_2 \cdot L_{\mathrm{E}} + \lambda_3 \cdot \mathrm{L2}(Q)$$
    1. Обновить Q
1. В течении T2 итераций:
    1. Получить из среды переход s, a, s', r, d
    2. Добавить в буфер (возможно, забыв старый опыт)
    3. Посчитать функцию потерь как в 2.2
    4. Обновить Q