

Policy gradient и что из него получается

Сергей Николенко

Академия MADE – Mail.Ru

11 ноября 2020 г.

Random facts:

- 11 ноября в Китае — День холостяков; дата была выбрана потому, что 11.11 символизирует людей, не состоящих в паре; самый популярный способ отметить — на обеде со своим единственным другом, но важно, чтобы каждый показал свою независимость
- 11 ноября 1572 г. Тихо Браге обнаружил вспышку сверхновой SN 1572 в созвездии Кассиопеи; по параллаксу Браге сделал вывод, что «новая звезда» находится намного дальше Луны
- 11 ноября 1842 г. в Пльзене был впервые подан новый сорт пива — Пильзнер
- 11 ноября 1983 г. калифорнийский студент Фред Коэн успешно завершил курсовую работу по созданию прототипа первого компьютерного вируса
- 11 ноября 2002 г. на arXiv появилась статья Григория Перельмана «The entropy formula for the Ricci flow and its geometric applications»

Policy gradient

Policy gradient

- Мы до сих пор говорили про action-value методы: берём марковский процесс принятия решений и ищем V и/или Q .
- Но это не единственный подход!
- Что если мы будем оптимизировать стратегии напрямую?

$$\pi(a|s, \theta) = p(A_t = a | S_t = s, \theta_t = \theta)$$

- Может быть, есть ещё и $\hat{V}(s, w)$, это отдельно.

Policy gradient

- Тогда можно попробовать сформулировать цель $J(\theta)$ и просто двигаться как

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta).$$

- Такие методы называются policy gradient методами.
- А если ещё и $\hat{V}(s, w)$, то actor-critic методами: критик – это \hat{V} , которая «критикует» стратегию π .

- Если действий не слишком много, можно искать $h(s, a, \theta)$ (хоть линейно, хоть нейросетью) и строить

$$\pi(a|s, \theta) = \text{softmax}(h).$$

- Преимущества:
 - можно сходиться к детерминированной стратегии без явной стратегии уменьшения ϵ ;
 - а можно строить стохастическую стратегию, которую через $Q(s, a)$ вообще непонятно как найти;
 - оптимизировать π может быть проще, чем Q .
- Но как же это сделать?

Policy gradient

- Policy gradient theorem:

$$\begin{aligned}\nabla V_\pi(s) &= \nabla \left(\sum_a \pi(a|s) Q_\pi(s, a) \right) = \\ &= \dots = \\ &= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} p(\text{перейти из } s \text{ в } x \text{ за } k \text{ шагов по } \pi) \sum_a \nabla \pi(a|x) Q_\pi(x, a),\end{aligned}$$

то есть

$$\nabla J(\theta) \propto \nabla V_\pi(s) = \sum_s \mu(s) \sum_a \nabla \pi(a|s) Q_\pi(s, a),$$

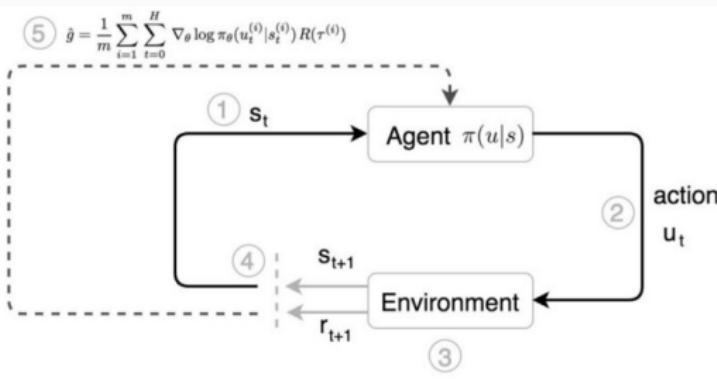
где $\mu(s)$ – ожидаемая доля времени, проведённого в состоянии s .

Policy gradient

- Иначе говоря,

$$\nabla J(\theta) \propto \mathbb{E}_\pi \left[\sum_a \nabla \pi(a|s) Q_\pi(s, a) \right],$$

и теперь можно и алгоритмы построить.



- Можно all-actions алгоритм:

$$\theta_{t+1} = \theta_t + \alpha \sum_a \hat{Q}(S_t, a, w) \nabla_\theta \pi(a|S_t, \theta).$$

Policy gradient

- Но лучше сразу уж REINFORCE (Williams, 1992):

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_\pi \left[\sum_a \nabla \pi(A_t | S_t, \boldsymbol{\theta}) Q_\pi(S_t, a) \right] = \\ &= \mathbb{E}_\pi \left[Q_\pi(S_t, A_t) \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta})}{\pi(A_t | S_t, \boldsymbol{\theta})} \right] = \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta})}{\pi(A_t | S_t, \boldsymbol{\theta})} \right],\end{aligned}$$

потому что $Q_\pi(S_t, A_t) = \mathbb{E}_\pi [G_t | S_t, A_t]$.

- И алгоритм такой:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla \log \pi(A_t | S_t, \boldsymbol{\theta}).$$

- Т.е. двигаем туда, где максимальный return, всё логично.

Policy gradient

- G_t можно оценить по методу Монте-Карло:

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Algorithm parameter: step size $\alpha > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|s, \theta)$

Loop for each step of the episode $t = 0, 1, \dots, T - 1$:

$$\begin{aligned} G &\leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \\ \theta &\leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t | S_t, \theta) \end{aligned} \tag{G_t}$$

Policy gradient

- Но это ещё не всё. Ещё можно добавить baseline:

$$\nabla J(\theta) \propto \nabla V_\pi(s) = \sum_s \mu(s) \sum_a (Q_\pi(s, a) - b(s)) \nabla \pi(a|s, \theta),$$

и это не повлияет ни на что, мы вычитаем

$$b(s) \nabla \sum_a \pi(a|s, \theta) = b(s) \nabla 1 = 0.$$

- Это не меняет ожидание, но может сильно изменить дисперсию! В том числе в хорошую сторону.
- Естественный baseline – это оценка состояния $\hat{V}(S_t, w)$.

Policy gradient

- Его нужно будет оценить, как мы обычно оцениваем \hat{V} , то есть теперь два разных градиента получается:

REINFORCE with Baseline (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Algorithm parameters: step sizes $\alpha^\theta > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot | \cdot, \theta)$

Loop for each step of the episode $t = 0, 1, \dots, T - 1$:

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \tag{G_t}$$

$$\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t, \mathbf{w})$$

$$\theta \leftarrow \theta + \alpha^\theta \gamma^t \delta \nabla \ln \pi(A_t | S_t, \theta)$$

Policy gradient

- А чтобы получился полноценный actor-critic, нужно через \hat{V} оценить и результаты действия тоже:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha \left(G_{t:t+1} - \hat{V}(S_t, w) \right) \nabla \log \pi(A_t | S_t, \boldsymbol{\theta}) = \\ &= \boldsymbol{\theta}_t + \alpha \left(R_{t+1} + \gamma \hat{V}(S_{t+1}, w) - \hat{V}(S_t, w) \right) \nabla \log \pi(A_t | S_t, \boldsymbol{\theta}) = \\ &= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta})}{\pi(A_t | S_t, \boldsymbol{\theta})}.\end{aligned}$$

Policy gradient

- Алгоритм получается такой:

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Parameters: step sizes $\alpha^\boldsymbol{\theta} > 0$, $\alpha^\mathbf{w} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

 Initialize S (first state of episode)

$I \leftarrow 1$

 Loop while S is not terminal (for each time step):

$A \sim \pi(\cdot|S, \boldsymbol{\theta})$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$ (if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^\mathbf{w} \delta \nabla \hat{v}(S, \mathbf{w})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^\boldsymbol{\theta} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$

$I \leftarrow \gamma I$

$S \leftarrow S'$

Policy gradient

- Можно с eligibility traces, как мы раньше обсуждали:

Actor-Critic with Eligibility Traces (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Parameters: trace-decay rates $\lambda^\theta \in [0, 1]$, $\lambda^w \in [0, 1]$; step sizes $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$ (e.g., to 0)

Loop forever (for each episode):

Initialize S (first state of episode)

$\mathbf{z}^\theta \leftarrow \mathbf{0}$ (d' -component eligibility trace vector)

$\mathbf{z}^w \leftarrow \mathbf{0}$ (d -component eligibility trace vector)

$$I \leftarrow 1$$

Loop while S is not terminal (for each time step):

$$A \sim \pi(\cdot | S, \theta)$$

Take action A , observe S', R

$$\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w}) \quad (\text{if } S' \text{ is terminal, then } \hat{v}(S', \mathbf{w}) \doteq 0)$$

$$\mathbf{z}^w \leftarrow \gamma \lambda^w \mathbf{z}^w + \nabla \hat{v}(S, w)$$

$$\mathbf{z}^\theta \leftarrow \gamma \lambda^\theta \mathbf{z}^\theta + I \nabla \ln \pi(A|S, \theta)$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$$

$$\theta \leftarrow \theta + \alpha^\theta \delta z^\theta$$

$$I \leftarrow \gamma I$$

$$S \uparrow S'$$

Policy gradient

- А можно попробовать сделать off-policy вариант; для другой стратегии β :

$$\begin{aligned}\nabla J(\theta) &\propto \nabla_{\theta} \sum_s \mu_{\pi}(s) \sum_a \pi(a|s) Q_{\pi}(s, a) = \\ &= \sum_s \mu_{\pi}(s) \sum_a [\nabla_{\theta} \pi(a|s) Q_{\pi}(s, a) + \pi(a|s) \nabla_{\theta} Q_{\pi}(s, a)],\end{aligned}$$

- Теперь первое слагаемое можно оценить как обычно:

$$\sum_s \mu_{\pi}(s)[...] = \mathbb{E}_{\pi} [...] = \mathbb{E}_{\beta} \left[\frac{\pi(a|s)}{\beta(a|s)} \dots \right], \text{ то есть}$$

$$\nabla J(\theta) \propto \mathbb{E}_{\beta} \left[\frac{\pi(a|s)}{\beta(a|s)} Q_{\pi}(s, a) \nabla_{\theta} \ln \pi(a|s) \right].$$

Policy gradient

- На первый взгляд кажется, что всё равно всё пропало, потому что $\nabla_{\theta} Q_{\pi}(s, a)$ мы никак не оценим.
- Но оказывается (Degris et al., 2012), что его можно просто выбросить, и полученная аппроксимация всё равно неплохая и минимум у неё там же.
- Так что в алгоритме просто добавятся importance weights:

Algorithm 1 The Off-PAC algorithm

Initialize the vectors \mathbf{e}_v , \mathbf{e}_u , and \mathbf{w} to zero

Initialize the vectors \mathbf{v} and \mathbf{u} arbitrarily

Initialize the state s

For each step:

Choose an action, a , according to $b(\cdot|s)$

Observe resultant reward, r , and next state, s'

$$\delta \leftarrow r + \gamma(s')\mathbf{v}^T \mathbf{x}_{s'} - \mathbf{v}^T \mathbf{x}_s$$

$$\rho \leftarrow \pi_{\mathbf{u}}(a|s)/b(a|s)$$

Update the critic ($\text{GTD}(\lambda)$ algorithm):

$$\mathbf{e}_v \leftarrow \rho (\mathbf{x}_s + \gamma(s)\lambda \mathbf{e}_v)$$

$$\mathbf{v} \leftarrow \mathbf{v} + \alpha_v [\delta \mathbf{e}_v - \gamma(s')(1 - \lambda)(\mathbf{w}^T \mathbf{e}_v) \mathbf{x}_s]$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_w [\delta \mathbf{e}_v - (\mathbf{w}^T \mathbf{x}_s) \mathbf{x}_s]$$

Update the actor:

$$\mathbf{e}_u \leftarrow \rho \left[\frac{\nabla_{\mathbf{u}} \pi_{\mathbf{u}}(a|s)}{\pi_{\mathbf{u}}(a|s)} + \gamma(s)\lambda \mathbf{e}_u \right]$$

$$\mathbf{u} \leftarrow \mathbf{u} + \alpha_u \delta \mathbf{e}_u$$

$$s \leftarrow s'$$

Policy gradient

- В policy gradient можно рассмотреть даже, например, непрерывные действия.
- Например, параметризуем действие $a \in \mathbb{R}$ через гауссиан:

$$\pi(a|s, \theta) = \frac{1}{\sigma(s, \theta)\sqrt{2\pi}} e^{-\frac{1}{2\sigma(s, \theta)}(a - \mu(s, \theta))^2}.$$

- Если, например,

$$\mu(s, \theta) = \theta_\mu^\top x_\mu(s), \quad \sigma(s, \theta) = e^{\theta_\sigma^\top x_\sigma(s)},$$

то можно подсчитать градиенты:

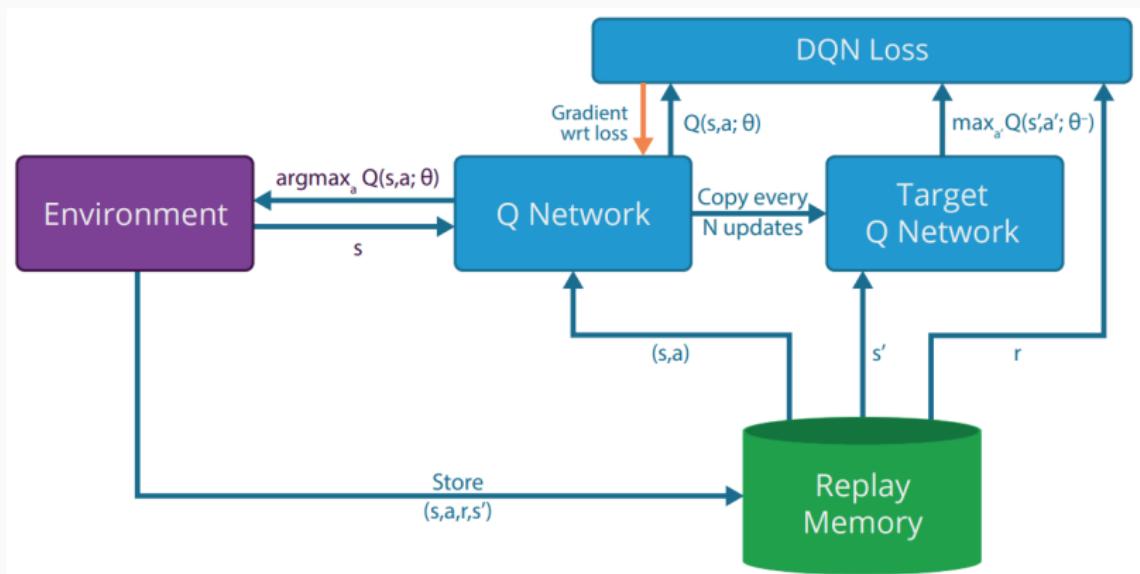
$$\nabla \ln \pi(a|s, \theta_\mu) = \frac{1}{\sigma(s, \theta)^2} (a - \mu(s, \theta)) x_\mu(s),$$

$$\nabla \ln \pi(a|s, \theta_\sigma) = \left(\frac{1}{\sigma(s, \theta)^2} (a - \sigma(s, \theta))^2 - 1 \right) x_\sigma(s).$$

Параллельные реализации RL и современные actor-critic алгоритмы

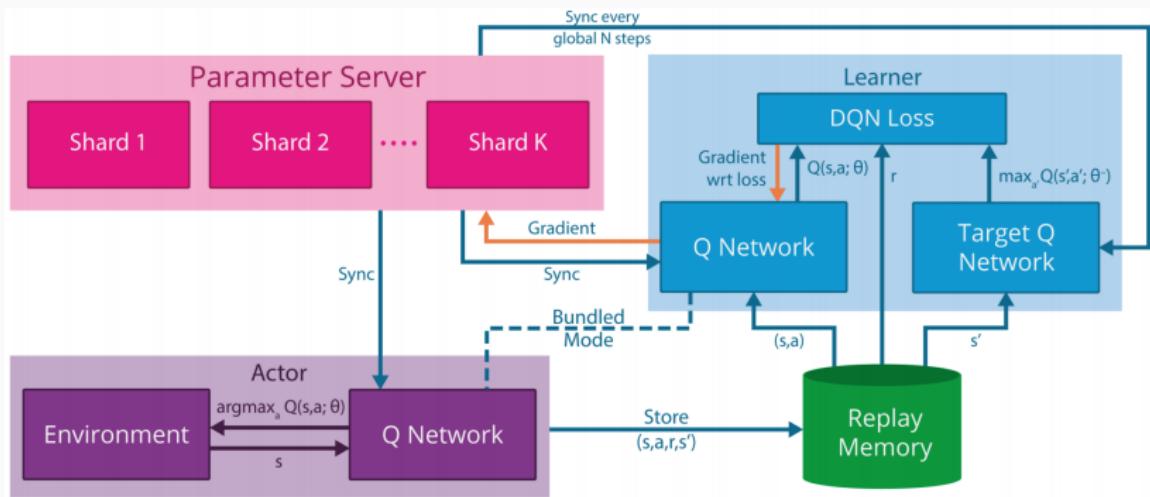
Современные actor-critic алгоритмы

- Вообще эта история тесно связана с тем, как распараллеливать RL.
- Вот что нам нужно сделать, например, в DQN:



Современные actor-critic алгоритмы

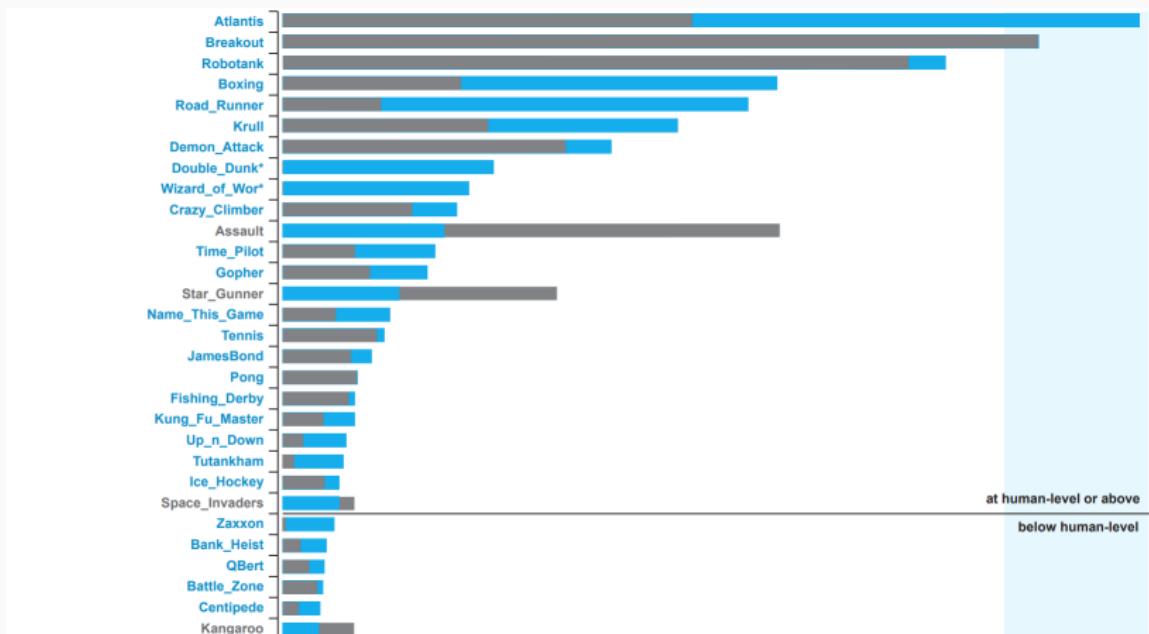
- (Nair, 2015), DeepMind: Gorila (General Reinforcement Learning Architecture)



- Параллелизуем на кучу агентов, храним всё в единой памяти, достаём оттуда для обучения, которое тоже параллелизовано и время от времени синхронизируется.

Современные actor-critic алгоритмы

- Достаточно типичная параллельная архитектура, улучшала базовый DQN от (Mnih et al., 2013) в большинстве случаев:



Современные actor-critic алгоритмы

- (Mnih et al., 2016), DeepMind: важная работа для actor-critic методов; во-первых, асинхронное DQN обучение, где просто накапливаются градиенты и иногда сбрасываются:

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

```
// Assume global shared  $\theta$ ,  $\theta^-$ , and counter  $T = 0$ .  
Initialize thread step counter  $t \leftarrow 0$   
Initialize target network weights  $\theta^- \leftarrow \theta$   
Initialize network gradients  $d\theta \leftarrow 0$   
Get initial state  $s$   
repeat  
    Take action  $a$  with  $\epsilon$ -greedy policy based on  $Q(s, a; \theta)$   
    Receive new state  $s'$  and reward  $r$   
     $y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$   
    Accumulate gradients wrt  $\theta$ :  $d\theta \leftarrow d\theta + \frac{\partial(y - Q(s, a; \theta))^2}{\partial \theta}$   
     $s = s'$   
     $T \leftarrow T + 1$  and  $t \leftarrow t + 1$   
    if  $T \bmod I_{target} == 0$  then  
        Update the target network  $\theta^- \leftarrow \theta$   
    end if  
    if  $t \bmod I_{AsyncUpdate} == 0$  or  $s$  is terminal then  
        Perform asynchronous update of  $\theta$  using  $d\theta$ .  
        Clear gradients  $d\theta \leftarrow 0$ .  
    end if  
until  $T > T_{max}$ 
```

Современные actor-critic алгоритмы

- Интереснее для нас – A3C (asynchronous advantage actor-critic):

- поддерживаем $\pi(A_t|S_t, \theta)$ и $V(S_t, w)$;
- обновляем

$$\nabla_{\theta'} \log \pi(a_t|s_t, \theta') A(s_t, a_t, \theta, \theta_v),$$

где $A(s_t, a_t, \theta, \theta_v)$ – это оценка advantage function, т.е.

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}, \theta_v) - V(s_t, \theta_v);$$

- обычно у сети softmax для $\pi(a_t|s_t, \theta)$ и ещё отдельно выход для $V(s_t, \theta_v)$;
- ещё хорошо бы добавить exploration, для этого мы добавляем в целевую функцию регуляризатор в виде энтропии π :

$$\nabla_{\theta'} \log \pi(a_t|s_t, \theta') (R_t - V(s_t, \theta_v)) + \beta \nabla_{\theta'} H(\pi(a_t|s_t, \theta')).$$

Современные actor-critic алгоритмы

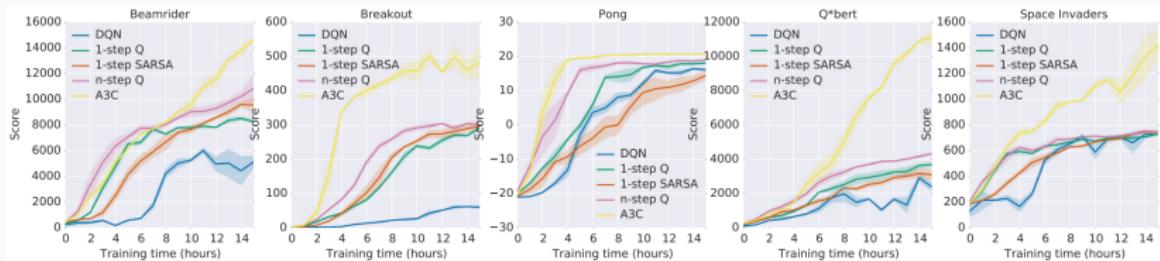
- Итого получается такой алгоритм:

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```
// Assume global shared parameter vectors  $\theta$  and  $\theta_v$  and global shared counter  $T = 0$ 
// Assume thread-specific parameter vectors  $\theta'$  and  $\theta'_v$ 
Initialize thread step counter  $t \leftarrow 1$ 
repeat
    Reset gradients:  $d\theta \leftarrow 0$  and  $d\theta_v \leftarrow 0$ .
    Synchronize thread-specific parameters  $\theta' = \theta$  and  $\theta'_v = \theta_v$ 
     $t_{start} = t$ 
    Get state  $s_t$ 
    repeat
        Perform  $a_t$  according to policy  $\pi(a_t|s_t; \theta')$ 
        Receive reward  $r_t$  and new state  $s_{t+1}$ 
         $t \leftarrow t + 1$ 
         $T \leftarrow T + 1$ 
    until terminal  $s_t$  or  $t - t_{start} == t_{max}$ 
     $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t; \theta'_v) & \text{for non-terminal } s_t \end{cases}$  // Bootstrap from last state
    for  $i \in \{t - 1, \dots, t_{start}\}$  do
         $R \leftarrow r_i + \gamma R$ 
        Accumulate gradients wrt  $\theta'$ :  $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta') (R - V(s_i; \theta'_v))$ 
        Accumulate gradients wrt  $\theta'_v$ :  $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$ 
    end for
    Perform asynchronous update of  $\theta$  using  $d\theta$  and of  $\theta_v$  using  $d\theta_v$ .
until  $T > T_{max}$ 
```

Современные actor-critic алгоритмы

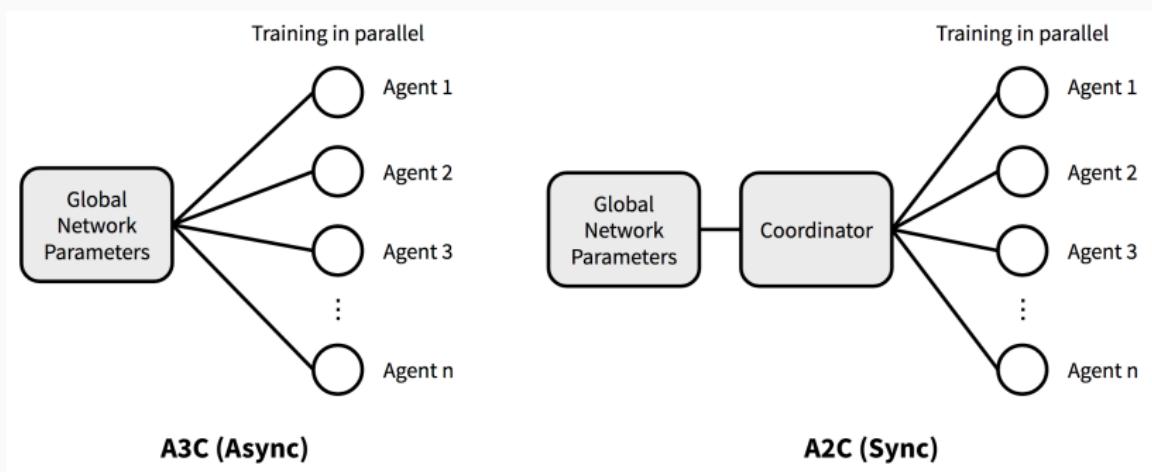
- И оказывается, что А3С побеждает всех подряд, да ещё и обучается быстрее:



Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Современные actor-critic алгоритмы

- Потом, правда, выяснилось, что A2C (то же самое, но без асинхронности) может работать даже лучше



Deterministic policy gradient

- Deterministic policy gradient (DPG): можно ли сделать то же самое с детерминированной стратегией $a = f(s; \theta)$?
- Кажется, что вряд ли, но вспомним, что целевая функция – это всё равно ожидание по состояниям, по времени $\mu_f(s)$, которое стратегия там проводит:

$$\theta_{k+1} = \theta_k + \alpha \mathbb{E}_{s \sim \mu_f} [\nabla_\theta Q_f(s, f(s; \theta))] .$$

- Возьмём градиент:

$$\theta_{k+1} = \theta_k + \alpha \mathbb{E}_{s \sim \mu_f} [\nabla_\theta f(s; \theta) \nabla_a Q_f(s, a) |_{a=f(s; \theta)}] .$$

Deterministic policy gradient

- И тогда можно сделать всё то же самое, доказать policy gradient теорему, off-policy actor-critic и т.д. (Silver et al., 2014)
- Например, такой алгоритм получится для on-policy actor-critic (Sarsa):

$$\delta_t = R_t + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t),$$

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q_w(s_t, a_t),$$

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta f(s_t, \theta) \nabla_a Q_w(s_t, a) |_{a=f_\theta(s)}$$

- А в off-policy даже и менять теперь ничего не надо, просто ожидание по β , а цель – это Q для стратегии f :

$$\delta_t = R_t + \gamma Q_w(s_{t+1}, f(s_{t+1}, \theta)) - Q_w(s_t, a_t),$$

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q_w(s_t, a_t),$$

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta f(s_t, \theta) \nabla_a Q_w(s_t, a) |_{a=f_\theta(s)}$$

Deterministic policy gradient

- Нейросети сюда применились в (Lillicrap et al., 2016): DDPG (Deep Deterministic Policy Gradient)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .

Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer R

for episode = 1, M **do**

 Initialize a random process \mathcal{N} for action exploration

 Receive initial observation state s_1

for t = 1, T **do**

 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise

 Execute action a_t and observe reward r_t and observe new state s_{t+1}

 Store transition (s_t, a_t, r_t, s_{t+1}) in R

 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

 Update the target networks:

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

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

end for

end for

Deterministic policy gradient

- (Barth-Maron et al., 2018): можно то же самое с распределениями сделать, D4PG (distributed distributional deep deterministic)

Algorithm 1 D4PG

Input: batch size M , trajectory length N , number of actors K , replay size R , exploration constant ϵ , initial learning rates α_0 and β_0

- 1: Initialize network weights (θ, w) at random
- 2: Initialize target weights $(\theta', w') \leftarrow (\theta, w)$
- 3: Launch K actors and replicate network weights (θ, w) to each actor
- 4: **for** $t = 1, \dots, T$ **do**
- 5: Sample M transitions $(\mathbf{x}_{i:i+N}, \mathbf{a}_{i:i+N-1}, r_{i:i+N-1})$ of length N from replay with priority p_i
- 6: Construct the target distributions $Y_i = \sum_{n=0}^{N-1} \gamma^n r_{i+n} + \gamma^N Z_{w'}(\mathbf{x}_{i+N}, \pi_{\theta'}(\mathbf{x}_{i+N}))$
- 7: Compute the actor and critic updates

$$\delta_w = \frac{1}{M} \sum_i \nabla_w (Rp_i)^{-1} d(Y_i, Z_w(\mathbf{x}_i, \mathbf{a}_i))$$
$$\delta_\theta = \frac{1}{M} \sum_i \nabla_\theta \pi_\theta(\mathbf{x}_i) \mathbb{E}[\nabla_\mathbf{a} Z_w(\mathbf{x}_i, \mathbf{a})] \Big|_{\mathbf{a}=\pi_\theta(\mathbf{x}_i)}$$

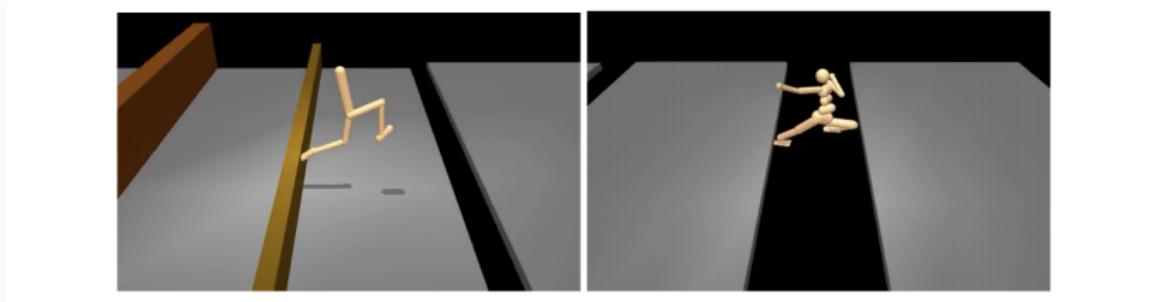
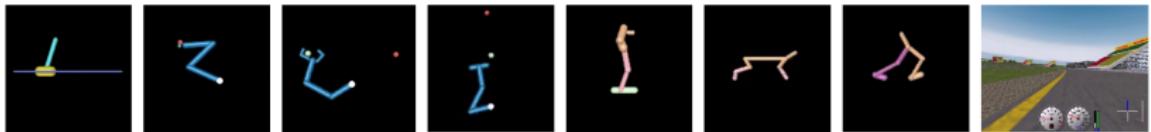
- 8: Update network parameters $\theta \leftarrow \theta + \alpha_t \delta_\theta$, $w \leftarrow w + \beta_t \delta_w$
- 9: If $t = 0 \bmod t_{\text{target}}$, update the target networks $(\theta', w') \leftarrow (\theta, w)$
- 10: If $t = 0 \bmod t_{\text{actors}}$, replicate network weights to the actors
- 11: **end for**
- 12: **return** policy parameters θ

Actor

- 1: **repeat**
- 2: Sample action $\mathbf{a} = \pi_\theta(\mathbf{x}) + \epsilon \mathcal{N}(0, 1)$
- 3: Execute action \mathbf{a} , observe reward r and state \mathbf{x}'
- 4: Store $(\mathbf{x}, \mathbf{a}, r, \mathbf{x}')$ in replay
- 5: **until** learner finishes

Deterministic policy gradient

- Это всё теперь, кстати, можно применять к роботике, т.е. к ситуациям с непрерывными действиями:



Deterministic policy gradient

- Получается хорошо:

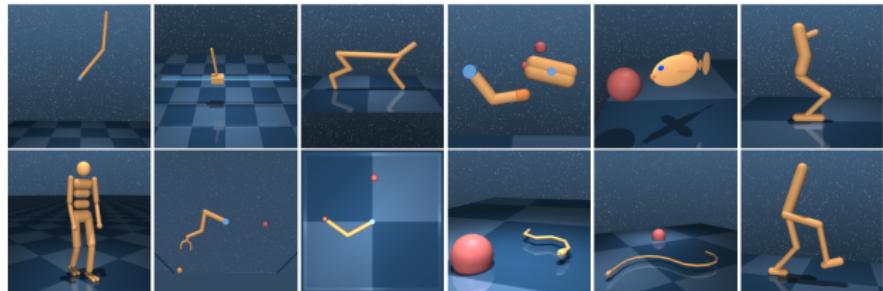
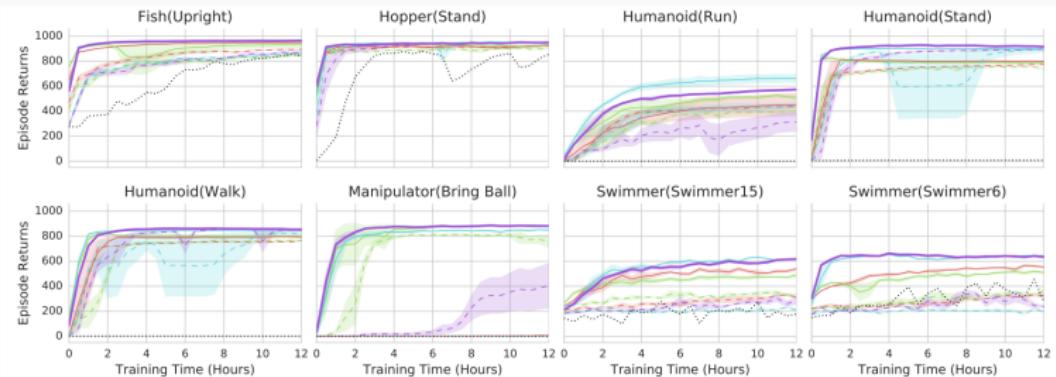


Figure 9: Control Suite domains used for benchmarking. *Top*: acrobot, cartpole, cheetah, finger, fish, hopper. *Bottom*: humanoid, manipulator, pendulum, reacher, swimmer6, swimmer15, walker.



Proximal policy optimization

- И последняя модификация: Proximal Policy Optimization (PPO)
- От OpenAI (Schulman et al., 2018):
 - обычно мы оптимизируем через автоматическое дифференцирование функции ошибки

$$L(\boldsymbol{\theta}) = \mathbb{E}_t [\log \pi(a_t | s_t, \boldsymbol{\theta}) A_t],$$

где A_t — оценка advantage function в момент времени t ;

- но если мы будем так вот напрямую оптимизировать, делая несколько шагов по данным с одной и той же траектории, ничего хорошего не получится;
- и вообще непонятно, как выбирать скорость обучения, как далеко идти и т.д.

Proximal policy optimization

- (Schulman et al., 2017): TRPO, Trust Region Policy Optimization
- Давайте оптимизировать

$$L(\boldsymbol{\theta}) = \mathbb{E}_t \left[\frac{\pi(a_t | s_t, \boldsymbol{\theta})}{\pi(a_t | s_t, \boldsymbol{\theta}_{\text{old}})} \log \pi(a_t | s_t, \boldsymbol{\theta}) A_t \right] = \mathbb{E}_t [r_t(\boldsymbol{\theta}) \log \pi(a_t | s_t, \boldsymbol{\theta}) A_t],$$

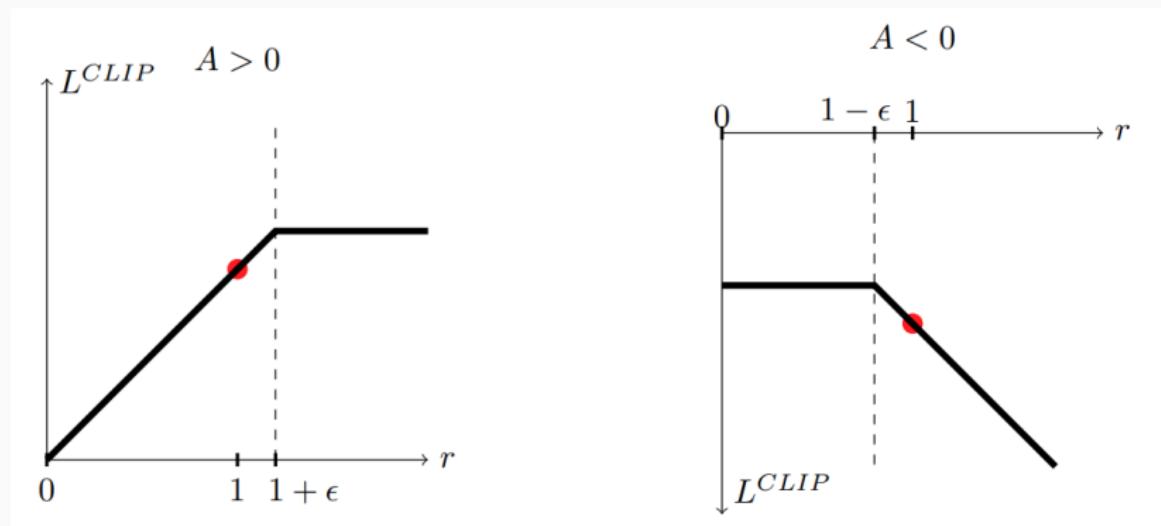
пока $\mathbb{E}_t [\text{KL}(\pi(\cdot | s_t, \boldsymbol{\theta}_{\text{old}}) \| \pi(\cdot | s_t, \boldsymbol{\theta}))] \leq \delta$.

- Условие можно заменить на регуляризатор, но трудно!
Значение регуляризатора будет меняться даже в процессе обучения одной задачи.
- А с условием хорошо работает, но нелегко реализовать вычислительно.

Proximal policy optimization

- PPO – давайте обрежем вместо регуляризатора:

$$L(\theta) = \mathbb{E}_t [\min (r_t(\theta) A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \log \pi(a_t | s_t, \theta) A_t)] .$$



Proximal policy optimization

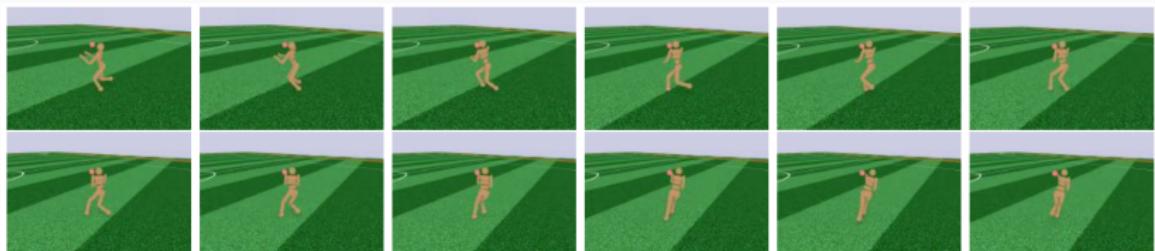
- Сам алгоритм теперь простой, считаем
 $A_t = -V(s_t) + r_t + \gamma r_{t+1} + \dots + \gamma^{T-t} V(s_T)$ и всё обучаем
распределённо:

Algorithm 1 PPO, Actor-Critic Style

```
for iteration=1, 2, ..., do
    for actor=1, 2, ..., N do
        Run policy  $\pi_{\theta_{\text{old}}}$  in environment for  $T$  timesteps
        Compute advantage estimates  $\hat{A}_1, \dots, \hat{A}_T$ 
    end for
    Optimize surrogate  $L$  wrt  $\theta$ , with  $K$  epochs and minibatch size  $M \leq NT$ 
     $\theta_{\text{old}} \leftarrow \theta$ 
end for
```

Proximal policy optimization

- Получается хорошо:



- <https://openai.com/blog/openai-baselines-ppo/>

Спасибо!

Спасибо за внимание!

