Reinforcement Learning

M2 Keisuke Fukuta

Agendas

1. Overview of RL
2. Hands On

Comparison of sample efficiency

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, etc.)

10x

model-based deep RL
(e.g. guided policy search)

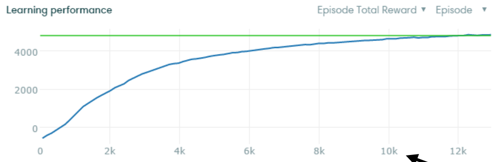
10x

model-based "shallow" RL
(e.g. PILCO)

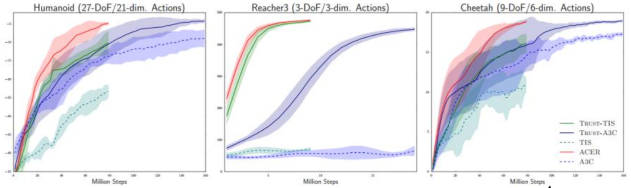
Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans[1] Jonathan Ho[1] Xi Chen[1] Ilya Sutskever[1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

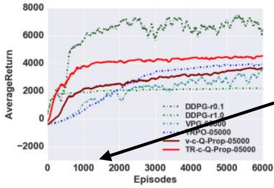


Wang et al. '17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

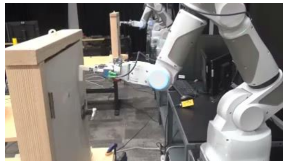
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)

half-cheetah

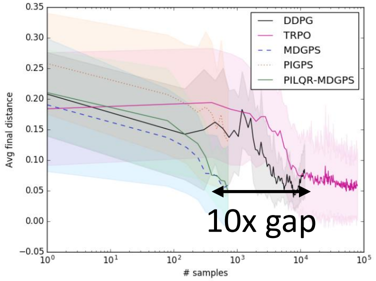


Gu et al. '16

1,000,000 steps
(1,000 episodes)
(~ 3 hours real time)



	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	$\approx 20\,s$	$\approx 60\,s$–$90\,s$	$\approx 20\,s$–$30\,s$
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$



Chebotar et al. '17 (note log scale)

about 20 minutes of experience on a real robot

Comparison of sample efficiency

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, etc.)

10x

model-based deep RL
(e.g. guided policy search)

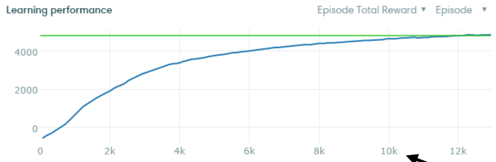
10x

model-based "shallow" RL
(e.g. PILCO)

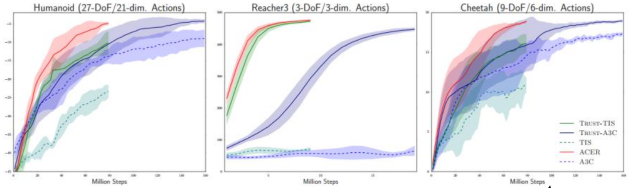
Evolution Strategies as a
Scalable Alternative to Reinforcement Learning

Tim Salimans [1] Jonathan Ho [1] Xi Chen [1] Ilya Sutskever [1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

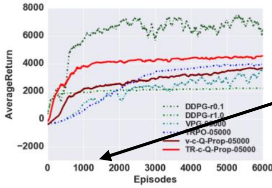


Wang et al. '17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

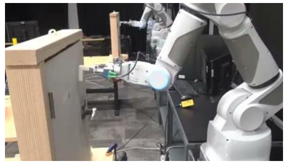
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)

half-cheetah

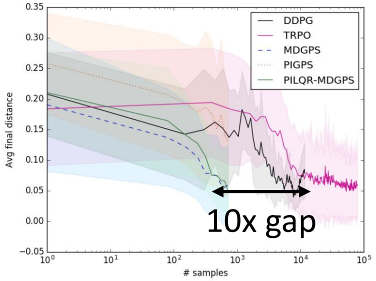


Gu et al. '16

1,000,000 steps
(1,000 episodes)
(~ 3 hours real time)



	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	$\approx 20\,s$	$\approx 60\,s$–$90\,s$	$\approx 20\,s$–$30\,s$
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$



Chebotar et al. '17 (note log scale)

about 20 minutes of
experience on a real
robot

Gradient free methods (Evolutionary Methods)

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}\left[\sum_{t=0}^{H} R(S_t|\pi_\theta)\right]$$

- General Idea
  - Update the parameters without using derivative of objective function

1. Make some random change to the parameters
2. If the result improves, keep the change. If not, discard the change.

E.g. Cross Entropy Method

CEM:

Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d_{>0}$

for iteration = 1, 2, ...

Sample n parameters $\theta_i \sim N(\mu, \mathrm{diag}(\sigma^2))$

For each θ_i, perform one rollout to get return $R(\tau_i)$

Select the top k% of θ, and fit a new diagonal Gaussian to those samples. Update μ, σ

endfor

Cross-Entropy Methodを用いたコンピュータ将棋における探索パラメータの学習;https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_uri&item_id=80931&file_id=1&file_no=1より

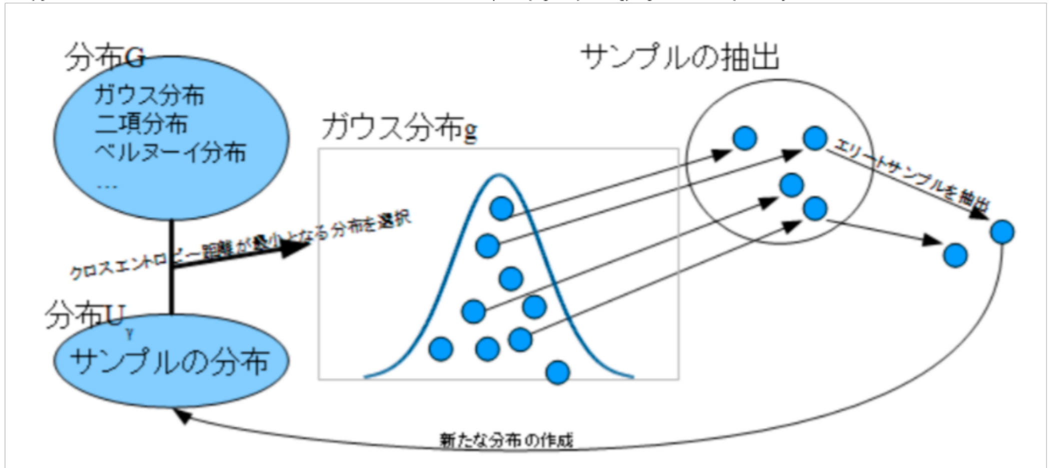


図 1 Cross-Entropy Method の概要図

Comparison of sample efficiency

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, etc.)

10x

model-based deep RL
(e.g. guided policy search)

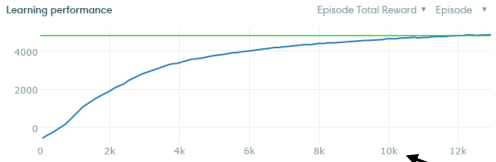
10x

model-based "shallow" RL
(e.g. PILCO)


Evolution Strategies as a Scalable Alternative to Reinforcement Learning

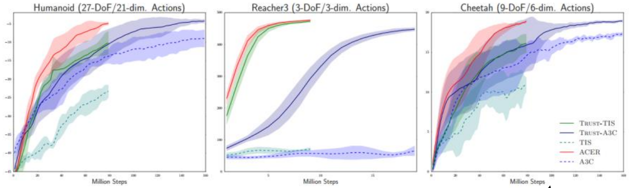
Tim Salimans[1] Jonathan Ho[1] Xi Chen[1] Ilya Sutskever[1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

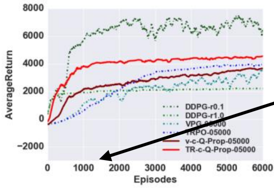
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)



Wang et al. '17

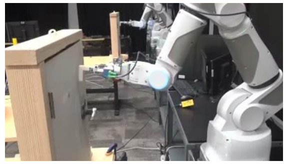
100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

half-cheetah

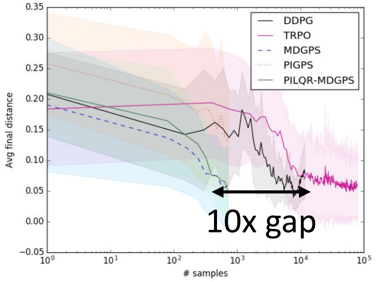


Gu et al. '16

1,000,000 steps
(1,000 episodes)
(~ 3 hours real time)



	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	$\approx 20\,s$	$\approx 60\,s$–$90\,s$	$\approx 20\,s$–$30\,s$
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$



Chebotar et al. '17 (note log scale)

about 20 minutes of experience on a real robot

Asynchronous Methods for Deep Reinforcement Learning [Mnih et al., 2016]

- 非同期に多数のエージェントを走らせてパラメータを同時に更新することでサンプル数を確保すると同時に入力の相関をなくすことができる

→ Experience Replayを使う必要がない

→ on-policyなRLアルゴリズムが使用可能!!

$$\nabla_\theta J(\pi_\theta) \cong \frac{1}{M} [\nabla_\theta \log \pi_\theta(a|s)\, r(s,a)]$$

- Advantage functionを用いたActor-Criticを非同期で走らせた結果、CPUで１日たった時点で他手法を大きく上回る
  (A3C : Asynchronous Advantage Actor Critic)

Comparison of sample efficiency

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, etc.)

10x

model-based deep RL
(e.g. guided policy search)

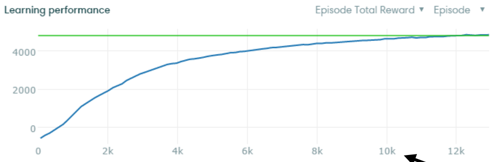
10x

model-based "shallow" RL
(e.g. PILCO)

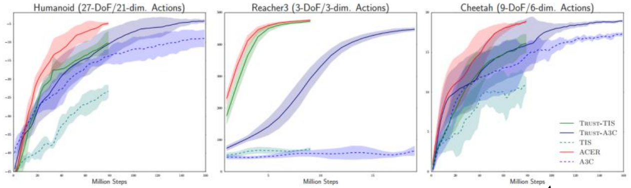
Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans [1] Jonathan Ho [1] Xi Chen [1] Ilya Sutskever [1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

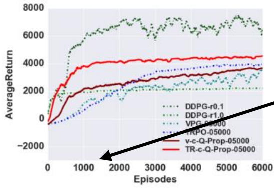


Wang et al. '17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

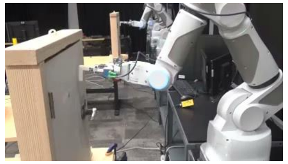
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)

half-cheetah

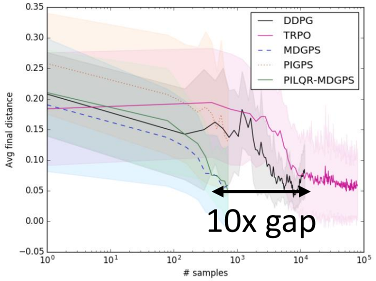


Gu et al. '16

1,000,000 steps
(1,000 episodes)
(~ 3 hours real time)



	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	$\approx 20\,s$	$\approx 60\,s$–$90\,s$	$\approx 20\,s$–$30\,s$
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$



Chebotar et al. '17 (note log scale)

about 20 minutes of experience on a real robot

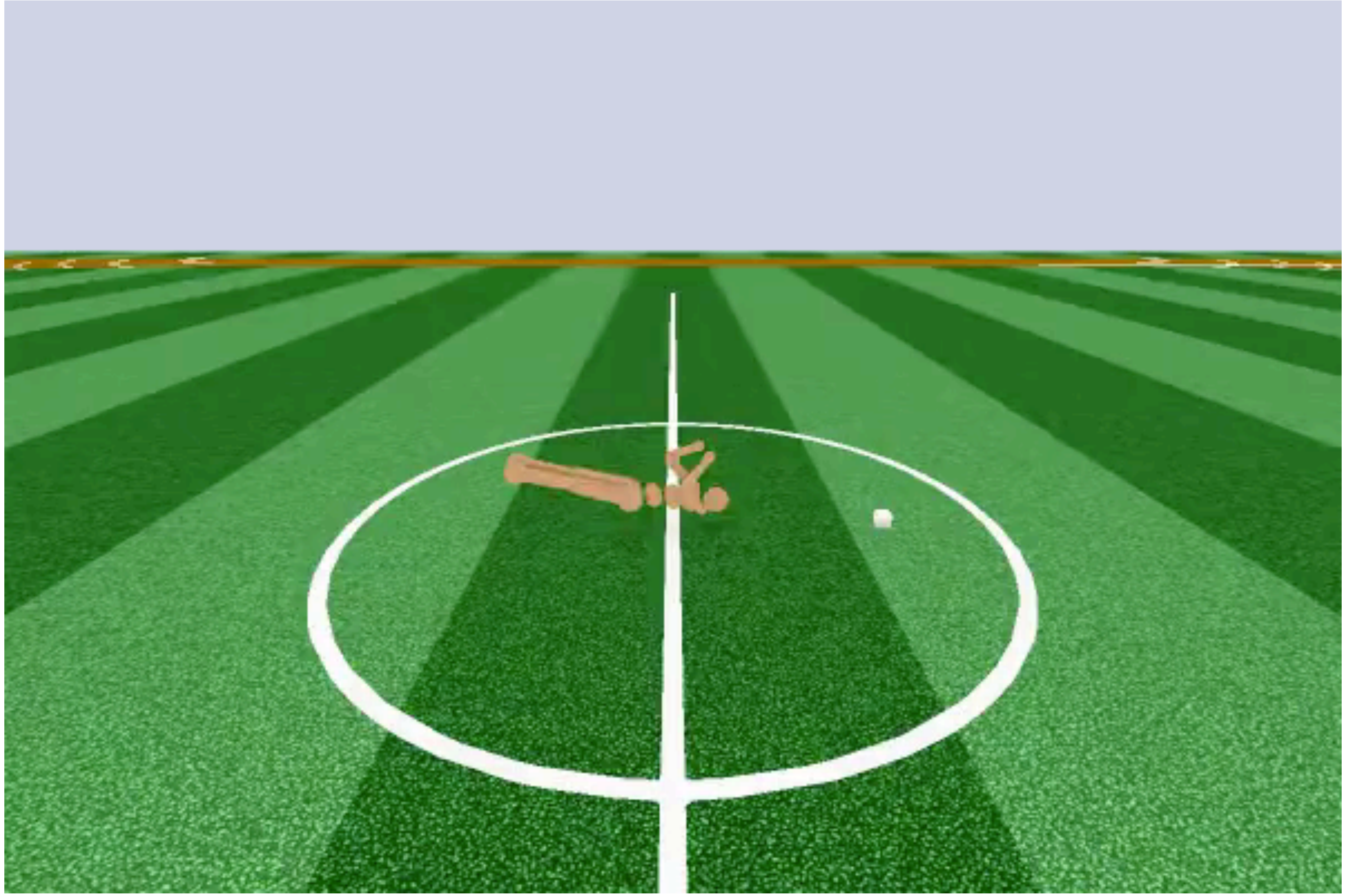
Trust Region Policy Optimization (TRPO)

- 最近流行りのアルゴリズム
- 概要
  - $D_{KL}(\theta_{old}, \theta) \leq \delta$ という範囲(Trust Region)において、期待コストを最小化するθを求める制約付き最適化問題を解くことで更新
  - Policyが大きく変わりすぎてほしくないけど、変わらなすぎも困る (learning rateの調整が難しい)という問題に対する一つの解
  - 結果的にNatural Policy gradientの亜種みたいな感じ

$$\underset{\theta}{\text{maximize}}\, \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}, a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)} Q_{\theta_{\text{old}}}(s,a)\right] \tag{14}$$

$$\text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s) \,\|\, \pi_\theta(\cdot|s))\right] \leq \delta.$$

Trust Region Policy Optimization (TRPO)



動画

Comparison of sample efficiency

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, etc.)

10x

model-based deep RL
(e.g. guided policy search)

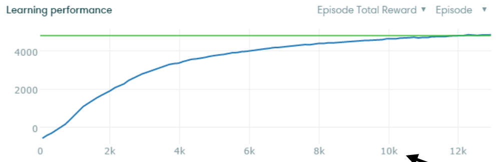
10x

model-based "shallow" RL
(e.g. PILCO)

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

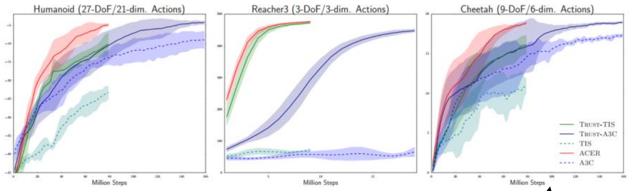
Tim Salimans[1] Jonathan Ho[1] Xi Chen[1] Ilya Sutskever[1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

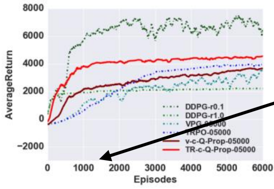
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)



Wang et al. '17

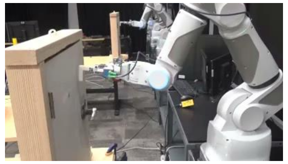
100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

half-cheetah

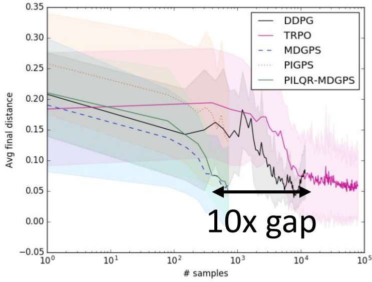


Gu et al. '16

1,000,000 steps
(1,000 episodes)
(~ 3 hours real time)



	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	$\approx 20\,s$	$\approx 60\,s$–$90\,s$	$\approx 20\,s$–$30\,s$
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$



Chebotar et al. '17 (note log scale)

about 20 minutes of
experience on a real
robot

Comparison of sample efficiency

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, etc.)

10x

model-based deep RL
(e.g. guided policy search)

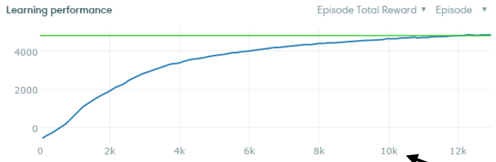
10x

model-based "shallow" RL
(e.g. PILCO)

Evolution Strategies as a
Scalable Alternative to Reinforcement Learning

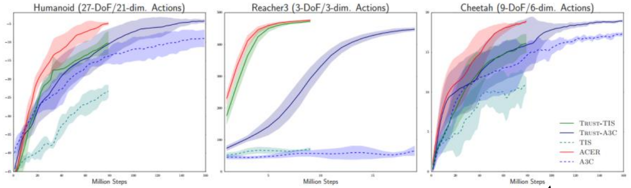
Tim Salimans [1] Jonathan Ho [1] Xi Chen [1] Ilya Sutskever [1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

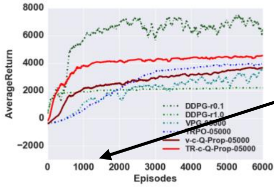
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)



Wang et al. '17

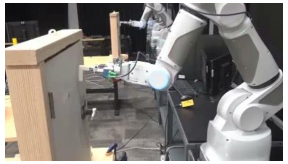
100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

half-cheetah

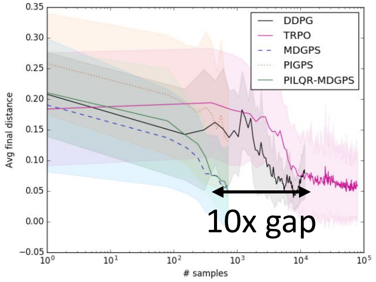


Gu et al. '16

1,000,000 steps
(1,000 episodes)
(~ 3 hours real time)



	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	≈ 20 s	≈ 60 s–90 s	≈ 20 s–30 s
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$



Chebotar et al. '17 (note log scale)

about 20 minutes of
experience on a real
robot

Model-based RL

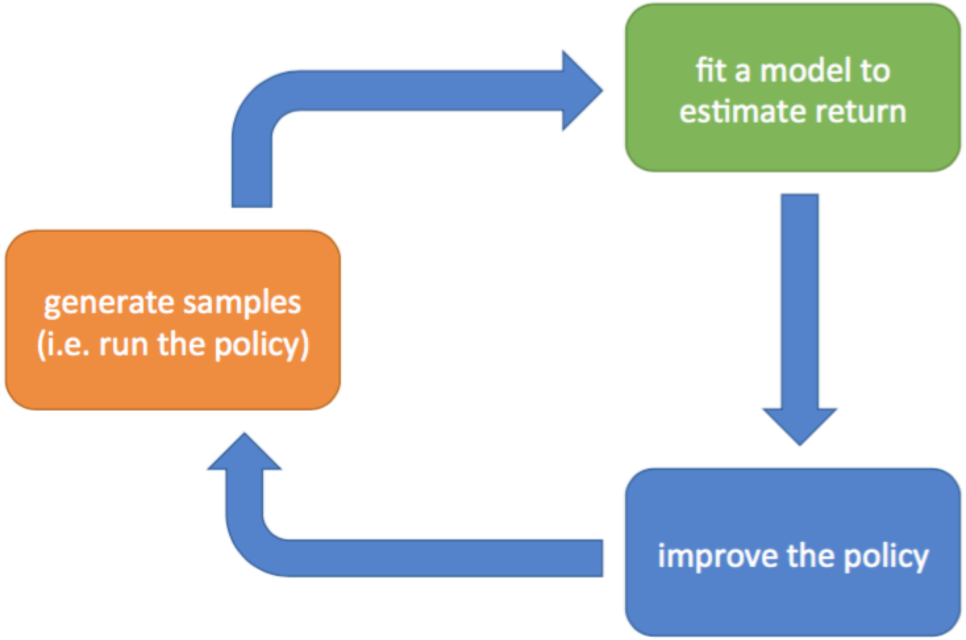
- So far….

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{policy}} \right]$$

Unknown

- Can we estimate transition function $P(s'|s,a)$? ? ? ? ?

Model-based RL



estimate $p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ using $f_\phi(\mathbf{s},\mathbf{a})$

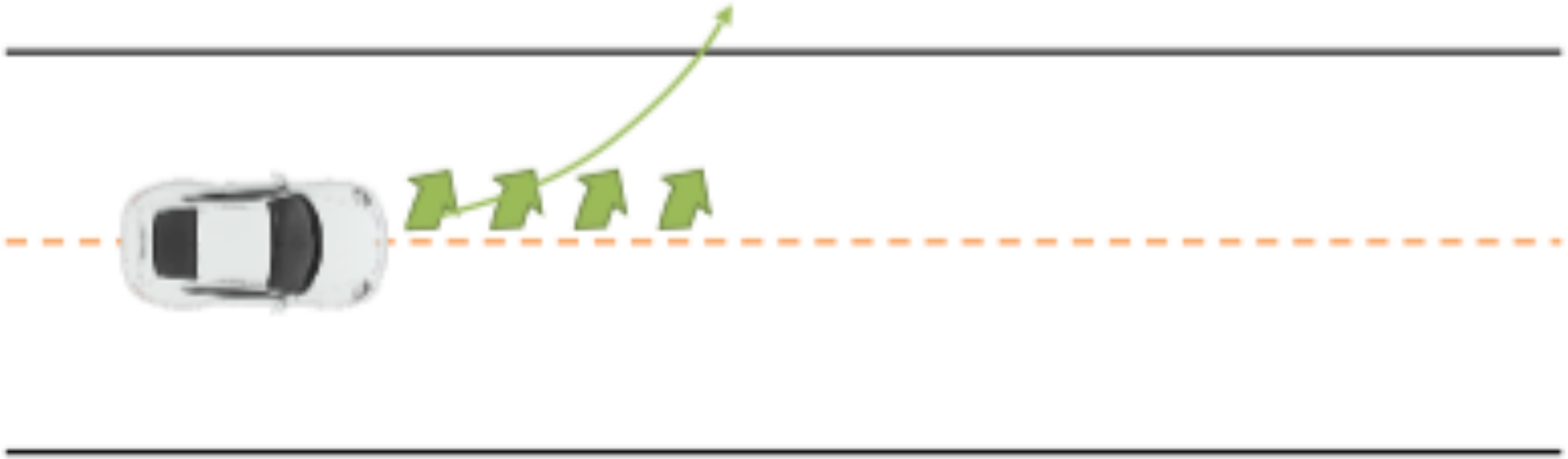
supervised learning

optimize $\pi_\theta(\mathbf{a}|\mathbf{s})$ (model-based)

e.g., backprop through model

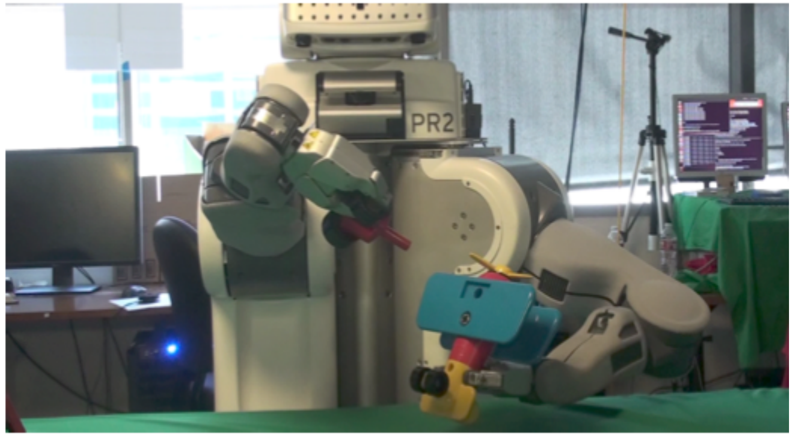
Model-based RL

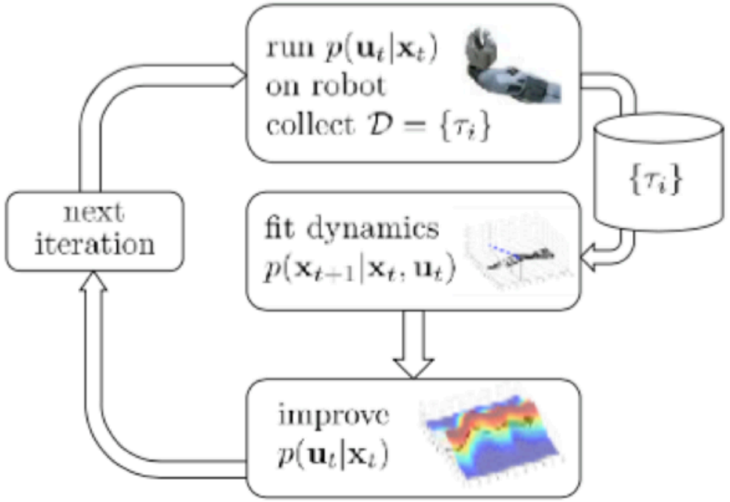
- Often fail to predict
- How to avoid it???



Guided Policy Search [Sergey Levine, 2015]

- 概要
  - UC Berkeleyのチーム
  - 実機ロボットでpolicyの学習
- 手法概要
  - 近傍のダイナミクスを線形近似し、
    ローカルに最適解を解析的に現代制御で解く。
  - Neural networkにそれぞれのローカルの解を転写





Guided Policy Search [Sergey Levine, 2015]

- https://www.youtube.com/watch?v=JeVppkoloXs

Comparison of sample efficiency

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, etc.)

10x

model-based deep RL
(e.g. guided policy search)

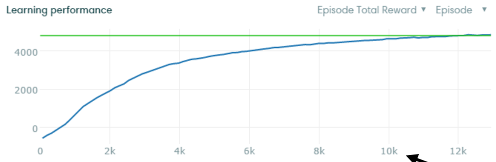
10x

model-based "shallow" RL
(e.g. PILCO)

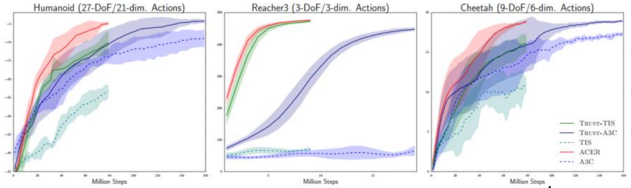
Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans[1] Jonathan Ho[1] Xi Chen[1] Ilya Sutskever[1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

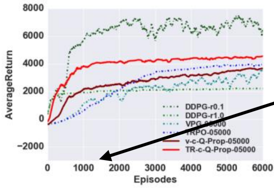


Wang et al. '17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

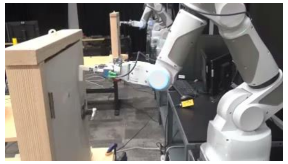
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)

half-cheetah

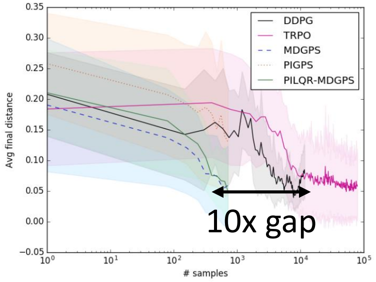


Gu et al. '16

1,000,000 steps
(1,000 episodes)
(~ 3 hours real time)



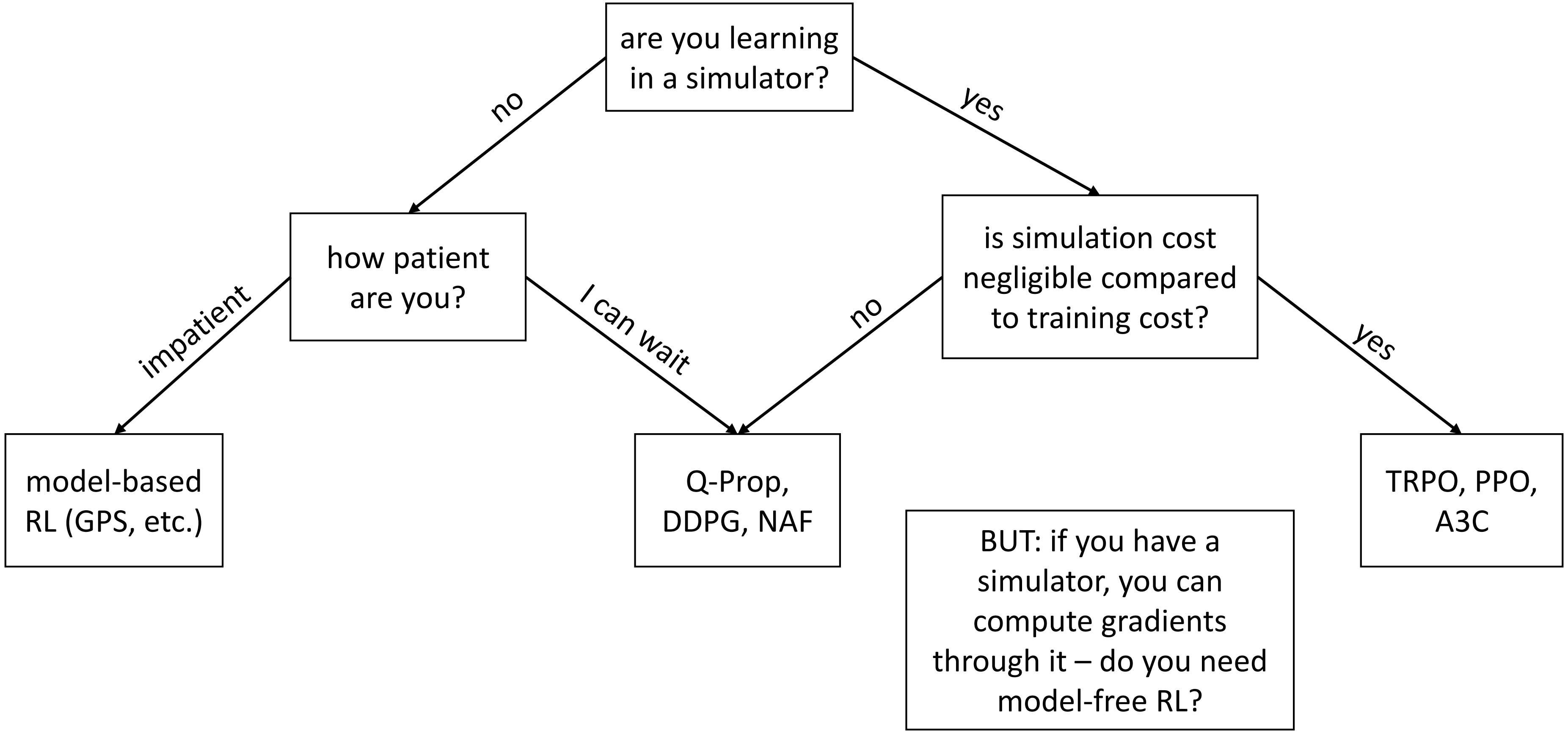
	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	≈ 20 s	≈ 60 s–90 s	≈ 20 s–30 s
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$



Chebotar et al. '17 (note log scale)

about 20 minutes of experience on a real robot

Which RL algorithm to use?

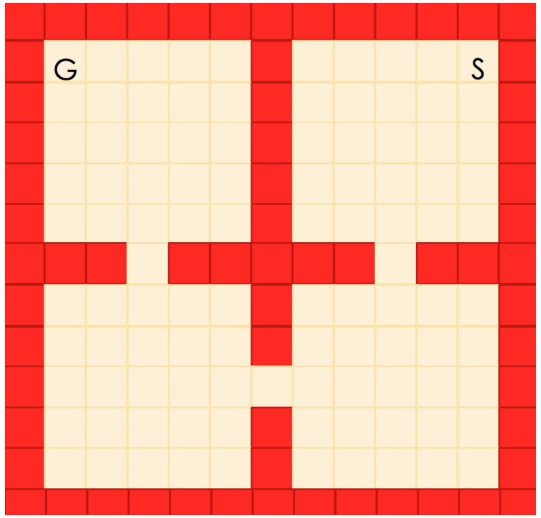


Design of Reward

- So far, "Environment" is given and fixed
  - E.g. state definition, Reward function
- E.g. Maze task
  - If reach the goal -> +100

    → It takes too long to time to get the goal randomly
  - Define Potential which get high if get close to the goal

    → It may reach the goal faster, but difficult to design such reward function increase hype-rparameters

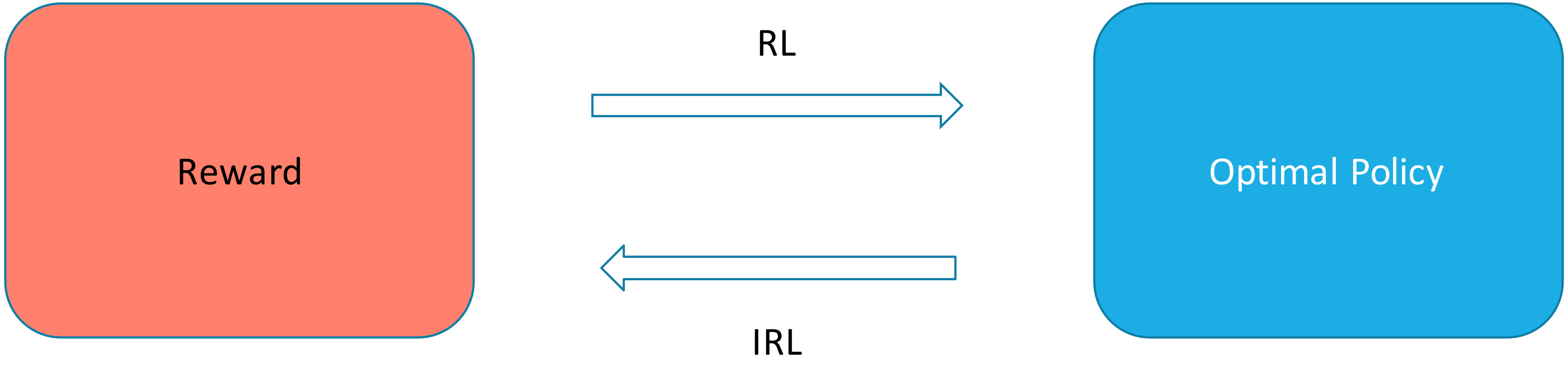


➔ Research about reward shaping

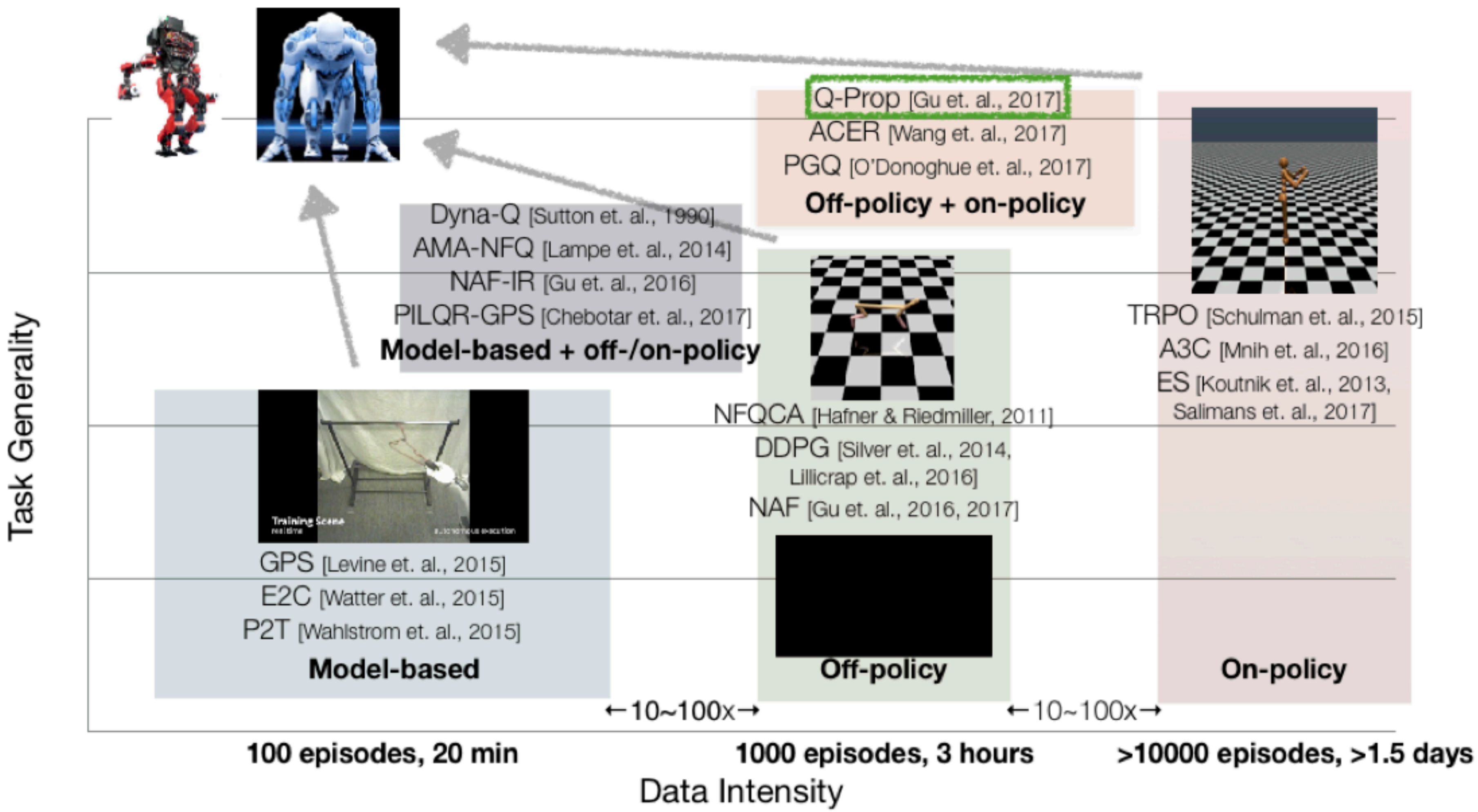
➔ Research about learning "reward function" $r(s, a)$

Inverse Reinforcement Learning

- In RL, reward or what is good -> optimal policy
- In IRL, optimal policy (human demonstration) -> reward function

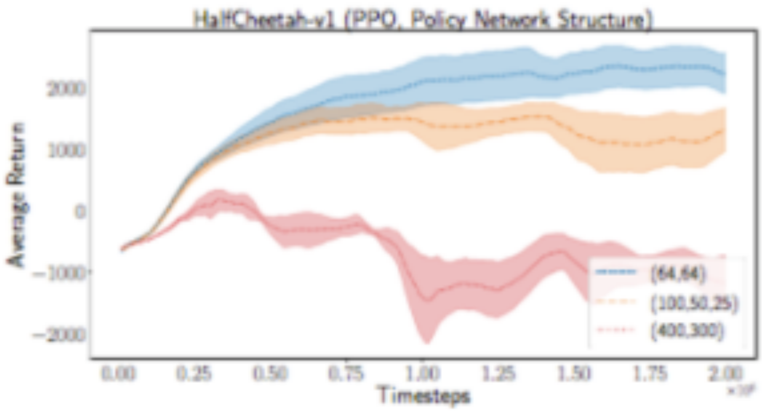


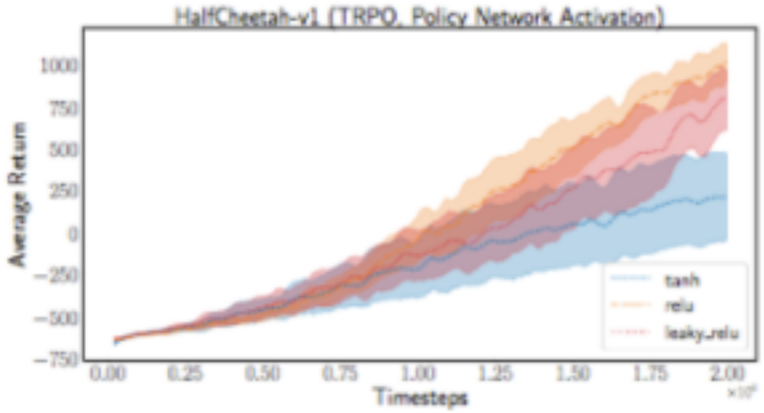
Deep RL in Robotics

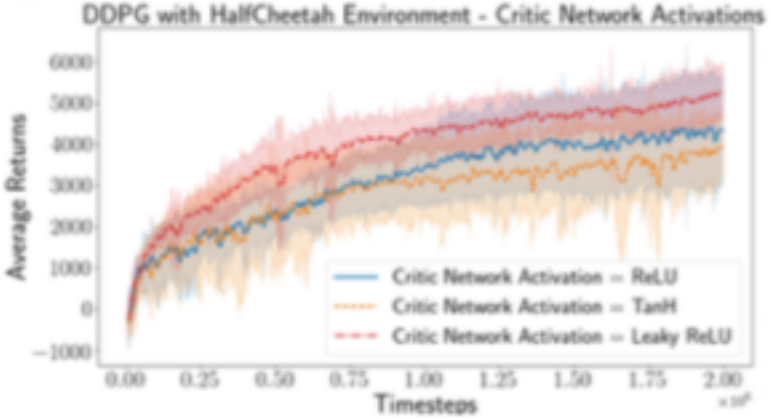


Darkness of RL

- RL is verrry sensitive to hyper parameters (even to seed)
- ↓ Different results from different activation function

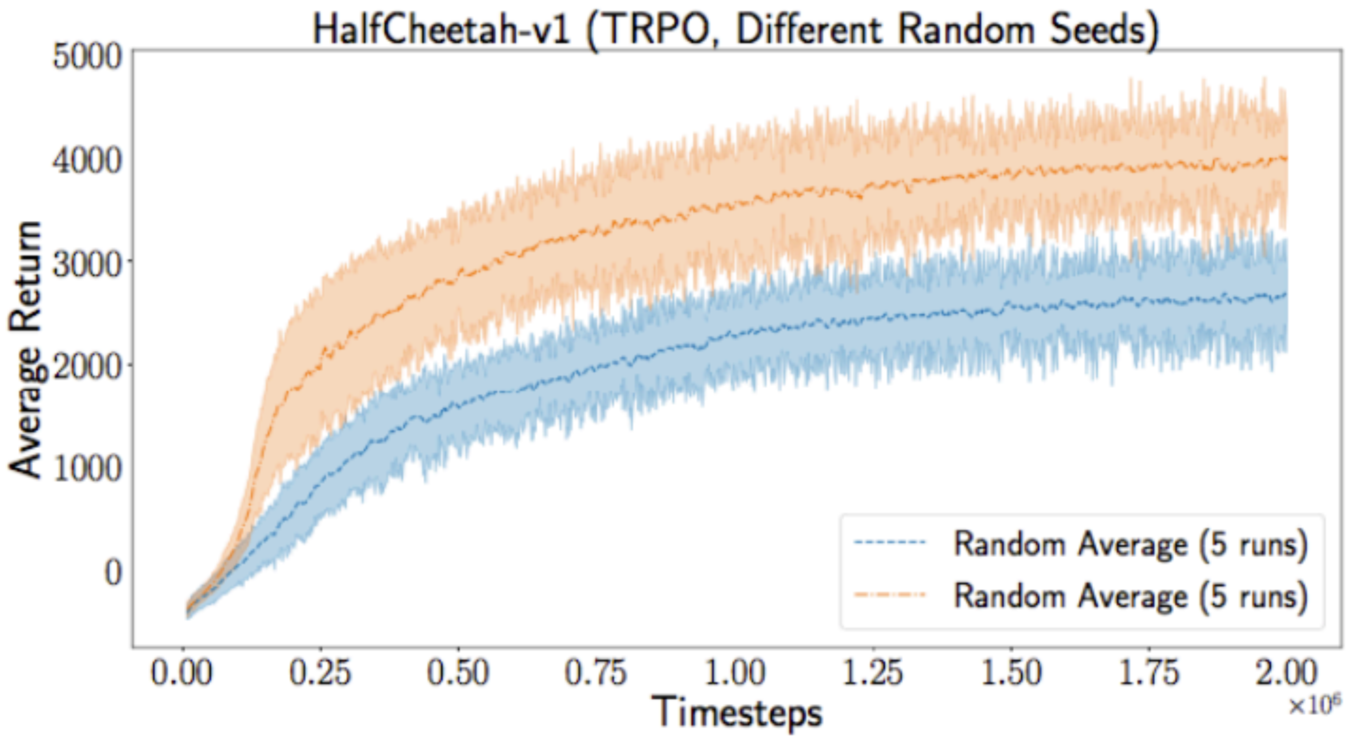






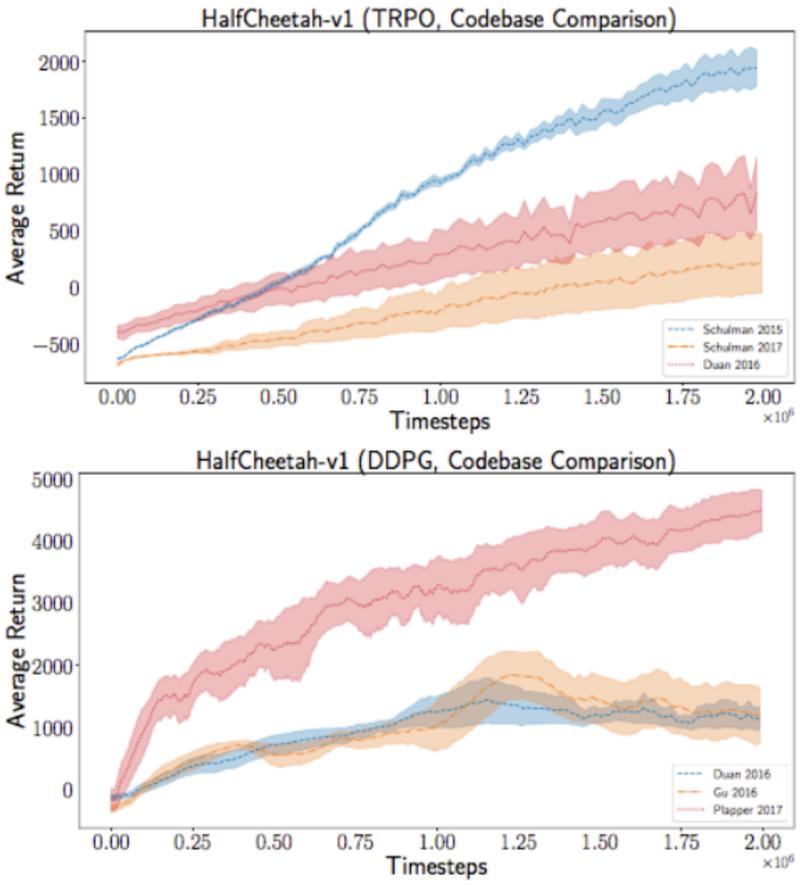
Darkness of RL

- Different results from different seeds



Darkness of RL

- Different results from different implementations



Summary

- RL is verrrry broad notion
  - Model-based or model-free
  - Value-based or Policy-based or Both
  - Off-policy or On-policy
- Recently, “Deep Reinforcement Learning” has been active domain of research
  - Interpret rich sensory inputs (DL)
  - Choose complex actions (RL)
- However, sample efficiency and robustness are still big problem

Reference

- David silverの講義資料
  - http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/intro_RL.pdf
- Policy search資料
  - http://icml.cc/2015/tutorials//PolicySearch.pdf
- DeepRLBootcamp資料
  - https://sites.google.com/view/deep-rl-bootcamp/labs
  - https://sites.google.com/view/deep-rl-bootcamp/lectures

おまけ

Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection
[Sergey Levine et al., 2016]

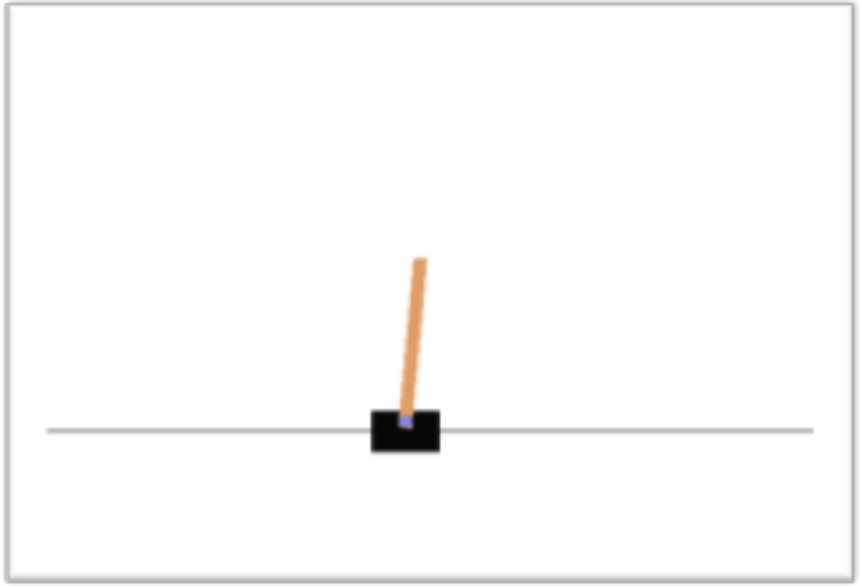
- Googleがロボットアーム14台並列に動かして
  物体のグラスピングを学習させてたやつ
- 概要
  1. 様々な状況でモーターを動かしてみて、成功したか失敗したかのサンプルを保存
  2. 保存したサンプルを利用して、Prediction Network $g(I_t, v_t)$ { I_t: 視覚情報, v_t: サーボへの命令 }を supervised-learning
  3. サーボへの命令は、Cross-Entropy-Methodで$g(I_t, v_t)$が高くなるvを探す
- 強化学習とよく言われるが、self-supervised learningと呼ばれる
  自分でデータサンプルを集めて学習する枠組み

Hands On

- We use “labs_final/” we provided last week
- We will focus on lab4 section 3. “Policy gradient” (mainly 3.6~)
- Learn card-pole agent with policy gradient!!!

We use

- “labs_final/lab4/simplepg/main.py”
- lab4.pdf (for material)



*Today we don’t use jupyter notebook

Hands On

- Lab4.pdf Section 3
  - **3.1 : background**
  - 3.2 : implementation for Point-v0 (optional)
  - 3.3 : implementation update function for Point-v0 (optional)
  - 3.4 : implementation baseline (optional)
  - 3.5 : implementation another baseline (optional)
  - **3.6 : implementation for CartPole-v0**
  - 3.7 : implementation natural gradient

Hands On

- Step1:
  - **Read 3.1 Background**
  - **understand policy gradient**

$$\nabla_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t r_t\right] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - b(s_t))\right]$$

- Step2:
  - **Read 3.6**
  - **Implement** ***cartpole_get_grad_logp_action() (main.py L94)***
  - **Implement** ***compute_update() (main.py L223)***
  - **Run command**

  $ ./docker_run.sh simplepg/main.py CartPole-v0 --use-baseline False --render True

Important!!!

Optional

- Step3: Read 3.2~3.4 and implement *compute_baselines()*

$ ./docker_run.sh simplepg/main.py CartPole-v0 --render True

- Step4: Read 3.7 and implement natural gradient

$ ./docker_run.sh simplepg/main.py CartPole-v0 --natural True --render True

* These answers are written in lab5

Hint

cartpole_get_grad_logp_action() (main.py L94)

A function, mapping from (theta, ob, action) to the gradient

- First, you have to add Add a constant term (1.0) to each observation

- Second, implement $\nabla_\theta \log \pi_\theta(a|s) = \left(e_a - \pi_\theta\,(\cdot\,|s)\right)\tilde{s}^T$
  - where e_a is a one-hot vector with all entries zero except in the a-th entry, where the value is 1.

Hint

compute_update() (main.py L223)

Function calculate policy gradient

$$\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - b(s_t))\right]$$

- First, calculate R_t from (R_tplus1, r_t)
- Second, multiply score function and advantage

Hint

cartpole_get_grad_logp_action()

- A function, mapping from (theta, ob, action) to the gradient (a matrix of size |A| * (|S|+1))
- First, you have to add Add a constant term (1.0) to each observation

```
ob_1 = include_bias(ob)
```

- Second, implement $\nabla_\theta \log \pi_\theta(a|s) = \left(e_a - \pi_\theta\,(\cdot\,|s)\right) s^T$
  - where e_a is a one-hot vector with all entries zero except in the ath entry, where the value is 1.

```
e_a = np.eye(theta.shape[0])[action]
logits = softmax(ob_1.dot(theta.T))
grad = np.outer(e_a - logits, ob_1)
return grad
```

Hint

compute_update() (main.py L223)

Function calculate policy gradient

$$\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - b(s_t))\right]$$

- First, calculate R_t from (R_tplus1, r_t)
- Second, multiply score function and advantage

```
R_t = discount * R_tplus1 + r_t
pg_theta = get_grad_logp_action(theta, s_t, a_t) * (R_t - b_t)
# pg_theta = np.zeros_like(theta)
"*** YOUR CODE HERE ***"
return R_t, pg_theta
```