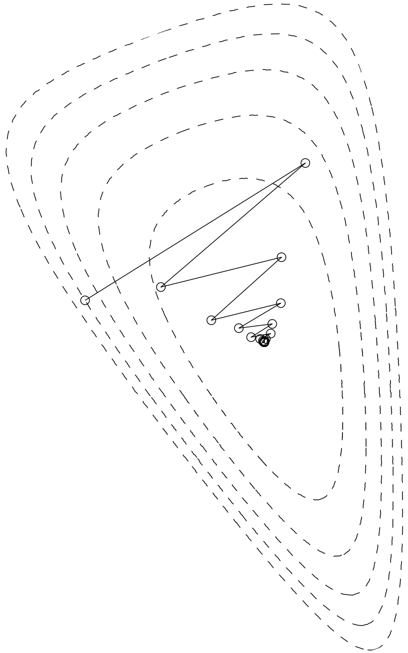


Optimization Algorithms

Reinforcement Learning & Optimization

Marc Toussaint
Technical University of Berlin
Winter 2024/25

- Reinforcement Learning is an optimization problem – how far can we get with standard optimization approaches rather than specialized RL methods?

Reinforcement Learning Basics

- *The world:* An MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P(s_{t+1} \mid s_t, a_t)$, reward fct $R(s_t, a_t)$, initial state distribution $P_0(s_0)$, and discounting factor $\gamma \in [0, 1]$.

Reinforcement Learning Basics

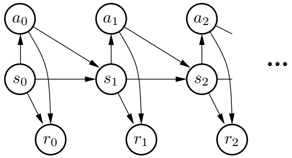
- *The world:* An MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P(s_{t+1} \mid s_t, a_t)$, reward fct $R(s_t, a_t)$, initial state distribution $P_0(s_0)$, and discounting factor $\gamma \in [0, 1]$.
- *The agent:* A policy $\pi(a_t | s_t)$.

Reinforcement Learning Basics

- *The world:* An MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P(s_{t+1} \mid s_t, a_t)$, reward fct $R(s_t, a_t)$, initial state distribution $P_0(s_0)$, and discounting factor $\gamma \in [0, 1]$.
- *The agent:* A policy $\pi(a_t|s_t)$.

- Together they define the path distribution $(\xi = (s_{0:T+1}, a_{0:T}))$

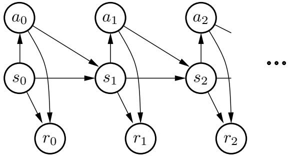
$$P_\pi(\xi) = P(s_0) \prod_{t=0}^{T} \pi(a_t|s_t)\ P(s_{t+1}|s_t, a_t)$$



Reinforcement Learning Basics

- *The world:* An MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P(s_{t+1} \mid s_t, a_t)$, reward fct $R(s_t, a_t)$, initial state distribution $P_0(s_0)$, and discounting factor $\gamma \in [0,1]$.
- *The agent:* A policy $\pi(a_t|s_t)$.

- Together they define the path distribution $(\xi = (s_{0:T+1}, a_{0:T}))$



$$P_\pi(\xi) = P(s_0) \prod_{t=0}^{T} \pi(a_t|s_t)\; P(s_{t+1}|s_t, a_t)$$

and the **expected total return**

$$J(\pi) = \mathbb{E}_{\xi \sim P_\pi} \Big\{ \underbrace{\textstyle\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)}_{R(\xi)} \Big\} = \int_\xi P_\pi(\xi)\; R(\xi)\; d\xi$$

Reinforcement Learning Basics

- We assume the policy $\pi_\theta(a|s)$ is parameterized by some $\theta \in \mathbb{R}^n$
- The problem is

$$\max_\theta J(\theta) \qquad \text{or} \qquad \max_\theta \int_\xi P_\theta(\xi)\ R(\xi)\ d\xi$$

Reinforcement Learning Basics

- We assume the policy $\pi_\theta(a|s)$ is parameterized by some $\theta \in \mathbb{R}^n$
- The problem is

$$\max_\theta J(\theta) \qquad \text{or} \qquad \max_\theta \int_\xi P_\theta(\xi)\ R(\xi)\ d\xi$$

  - $J(\theta)$ is just a function we want to optimize
  - $J(\theta)$ is a "weighted sum" over all paths (cf. additive cost function & SGD)
  - We can't really compute/evaluate $f(\theta)$ exactly – we can only get a sample $\xi \sim P_\theta$ and $R(\xi)$ in each iteration (cf. SGD case!)
  - Different: $\sum_i f_i(x) \leftrightarrow$ fixed distribution over i; $\int_\xi P_\theta(\xi)R(\xi) \leftrightarrow$ non-stationary distribution over ξ

Reinforcement Learning Basics

- We assume the policy $\pi_\theta(a|s)$ is parameterized by some $\theta \in \mathbb{R}^n$
- The problem is

$$\max_\theta J(\theta) \qquad \text{or} \qquad \max_\theta \int_\xi P_\theta(\xi)\ R(\xi)\ d\xi$$

  - $J(\theta)$ is just a function we want to optimize
  - $J(\theta)$ is a “weighted sum” over all paths (cf. additive cost function & SGD)
  - We can’t really compute/evaluate $f(\theta)$ exactly – we can only get a sample $\xi \sim P_\theta$ and $R(\xi)$ in each iteration (cf. SGD case!)
  - Different: $\sum_i f_i(x) \leftrightarrow$ fixed distribution over i; $\int_\xi P_\theta(\xi)R(\xi) \leftrightarrow$ non-stationary distribution over ξ

Can knowing about the MDP process simplify the optimization problem?

Can knowing about the MDP process simplify the optimization problem?

- Yes, in at least 2 ways:

- Bellman optimality – we understand sth. about the optimal policy beyond KKT
- Policy gradients – we can derive gradients

Bellman optimality condition

- In general optimization, optima x^* are only characterized by KKT, or stationarity

Bellman optimality condition

- In general optimization, optima x^* are only characterized by KKT, or stationarity
- When $\max_\theta J(\theta)$, we know another *condition of optimality*: **Bellman optimality**
  - The value function $V^*(x)$ over state space fulfills

$$V^*(s) = \max_a \Big[R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{V^*(s')\} \Big]$$

  - Knowing that function implies the optimal policy π^*

Bellman optimality condition

- In general optimization, optima x^* are only characterized by KKT, or stationarity
- When $\max_\theta J(\theta)$, we know another *condition of optimality*: **Bellman optimality**
  - The value function $V^*(x)$ over state space fulfills

$$V^*(s) = \max_a \Big[R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{V^*(s')\} \Big]$$

  - Knowing that function implies the optimal policy π^*
  - But that also raises a problem! If π_θ is parameteric! And/or $V(s)$ is parameteric! We raise extra function approximation problems. Read:

    Lagoudakis & Parr: *Least-squares policy iteration*. JMLR 2003

Bellman optimality condition

- In general optimization, optima x^* are only characterized by KKT, or stationarity
- When $\max_\theta J(\theta)$, we know another *condition of optimality*: **Bellman optimality**
  - The value function $V^*(x)$ over state space fulfills

$$V^*(s) = \max_a \Big[R(s,a) + \gamma \mathbb{E}_{s'|s,a} \big\{ V^*(s') \big\} \Big]$$

  - Knowing that function implies the optimal policy π^*
  - But that also raises a problem! If π_θ is parameteric! And/or $V(s)$ is parameteric! We raise extra function approximation problems. Read:

    Lagoudakis & Parr: *Least-squares policy iteration*. JMLR 2003

- The Bellman optimality condition truely exploits the MDP structure, and gives further conditions on the optimum beyond stationarity of $J(\theta)$.

- Learning (=optimizing) while collecting more and more data
  - Unusual from the optimization perspective $\leftrightarrow$ instable “target” (objective, ξ-distribution)
  - Leads to breadth of RL-methodologies (model-based/model-free RL, TD-, Q-learning, etc)

- Learning (=optimizing) while collecting more and more data
  - Unusual from the optimization perspective $\leftrightarrow$ instable “target” (objective, ξ-distribution)
  - Leads to breadth of RL-methodologies (model-based/model-free RL, TD-, Q-learning, etc)

- But there are also trends to avoid this
  - “Offline RL”, classical system identification, model-based RL
  - separating data collection issue from optimization issue

Stochastic Policy Gradient

- Recall

$$J(\theta) = \int_\xi P_\theta(\xi)\ R(\xi)\ d\xi$$

- We have

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \int P_\theta(\xi)\ R(\xi)\ d\xi = \int P_\theta(\xi) \nabla_\theta \log P_\theta(\xi) R(\xi) d\xi \\ &= \mathbb{E}_{\xi|\theta}\{\nabla_\theta \log P_\theta(\xi) R(\xi)\} = \mathbb{E}_{\xi|\theta}\Big\{ \textstyle\sum_{t=0}^H \nabla_\theta \log \pi(a_t|s_t) R(\xi) \Big\} \\ &= \mathbb{E}_{\xi|\theta}\Big\{ \textstyle\sum_{t=0}^H \nabla_\theta \log \pi(a_t|s_t)\ \gamma^t \underbrace{\textstyle\sum_{t'=t}^H \gamma^{t'-t} r_{t'}}_{\hat{Q}_{\xi,t}} \Big\}\end{aligned}$$

Deterministic Policy Gradient

- However, in practise, policies are often not stochastic. Esp. neural networks. We have $a = \pi_\theta(s) \in \mathbb{R}^d$, parameterized by θ. What is the correct gradient then?
- As introduced in reference [2]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim P_\theta}\left\{ \nabla_\theta \pi_\theta(s)\ \nabla_a Q^{\pi_\theta}(s,a)\Big|_{a=\pi_\theta(s)} \right\}$$

(NOTE: unusual convention about Jacobians... I'd write it $\partial_a Q^{\pi_\theta}(s,a)\partial_\theta \pi(s)$)

Silver et al: *Deterministic policy gradient algorithms*. 2014

Deterministic Policy Gradient

- However, in practise, policies are often not stochastic. Esp. neural networks. We have $a = \pi_\theta(s) \in \mathbb{R}^d$, parameterized by θ. What is the correct gradient then?
- As introduced in reference [2]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim P_\theta} \Big\{ \nabla_\theta \pi_\theta(s) \; \nabla_a Q^{\pi_\theta}(s,a) \big|_{a=\pi_\theta(s)} \Big\}$$

(NOTE: unusual convention about Jacobians... I'd write it $\partial_a Q^{\pi_\theta}(s,a) \partial_\theta \pi(s)$)

Silver et al: *Deterministic policy gradient algorithms*. 2014

- So we in principle also have a gradient! But very noisy! Better: D4PG

Conclusions

- **Can knowing about the MDP process simplify the optimization problem?** Yes:
  - Bellman optimality, gradients
  - interleaved learning/optimization and data collection
  - Esp. if “reward signal” is informative beyond total return (dense rewards)

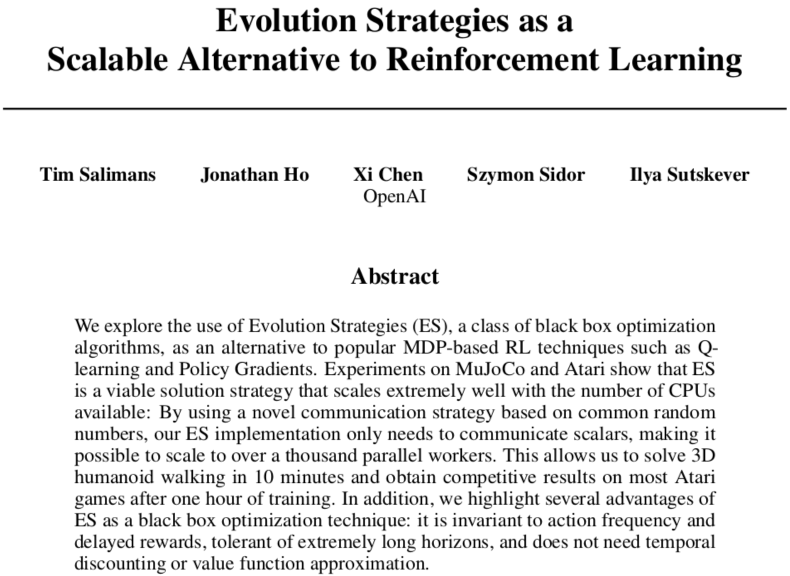
Conclusions

- **Can knowing about the MDP process simplify the optimization problem?** Yes:
  - Bellman optimality, gradients
  - interleaved learning/optimization and data collection
  - Esp. if “reward signal” is informative beyond total return (dense rewards)

- **However,** reasons to ignore structure of underlying MPD:
  - Avoid implied problems, e.g. by function approximation, value estimation, policy iteration
  - very noisy gradient estimates
  - Robustness to mis-assumptions
- $\rightarrow$ black-box or **derivative-free optimization**

References

- Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864.
- Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint arXiv:1712.06567.
- Stulp, F., & Sigaud, O. (2013). Robot skill learning: From reinforcement learning to evolution strategies. Paladyn, Journal of Behavioral Robotics, 4(1), 49-61.



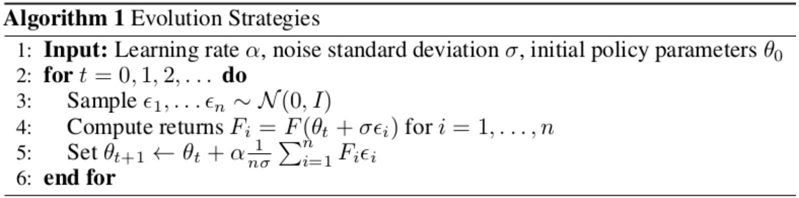
Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans **Jonathan Ho** **Xi Chen** **Szymon Sidor** **Ilya Sutskever**
OpenAI

Abstract

We explore the use of Evolution Strategies (ES), a class of black box optimization algorithms, as an alternative to popular MDP-based RL techniques such as Q-learning and Policy Gradients. Experiments on MuJoCo and Atari show that ES is a viable solution strategy that scales extremely well with the number of CPUs available: By using a novel communication strategy based on common random numbers, our ES implementation only needs to communicate scalars, making it possible to scale to over a thousand parallel workers. This allows us to solve 3D humanoid walking in 10 minutes and obtain competitive results on most Atari games after one hour of training. In addition, we highlight several advantages of ES as a black box optimization technique: it is invariant to action frequency and delayed rewards, tolerant of extremely long horizons, and does not need temporal discounting or value function approximation.

- The ES they employ:

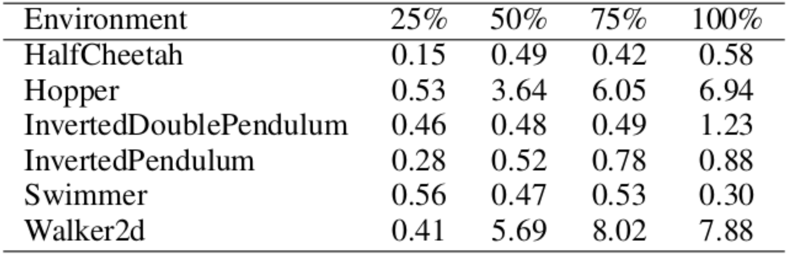


Algorithm 1 Evolution Strategies

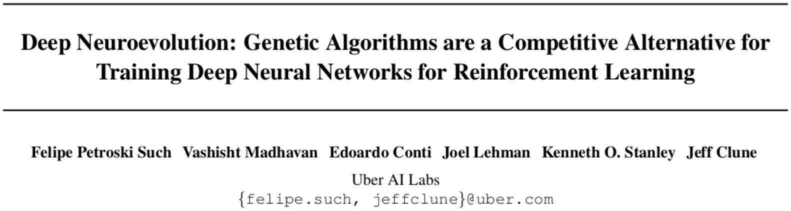
```
1: Input: Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$
2: for $t = 0, 1, 2, \ldots$ do
3:     Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
4:     Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
5:     Set $\theta_{t+1} \leftarrow \theta_t + \alpha\frac{1}{n\sigma}\sum_{i=1}^{n} F_i\epsilon_i$
6: end for
```

  - Is an instance of our "General Stochastic Search" scheme:
    θ is the mean; $\lambda = n$ samples $x_t \sim \mathcal{N}(\theta, \sigma^2 I)$; evaluations $f(x_i)$
  - The update is more like a stochastic gradient step rather than selection
    In expectation, $F_i\epsilon_i \doteq (F_\theta + \nabla F^\top \sigma\epsilon_i)\epsilon_i \approx 0 + \sigma\nabla F$.

- Ratio of ES timesteps to TRPO timesteps needed to reach various percentages of TRPO's learning progress at 5 million timesteps:



Environment	25%	50%	75%	100%
HalfCheetah	0.15	0.49	0.42	0.58
Hopper	0.53	3.64	6.05	6.94
InvertedDoublePendulum	0.46	0.48	0.49	1.23
InvertedPendulum	0.28	0.52	0.78	0.88
Swimmer	0.56	0.47	0.53	0.30
Walker2d	0.41	5.69	8.02	7.88



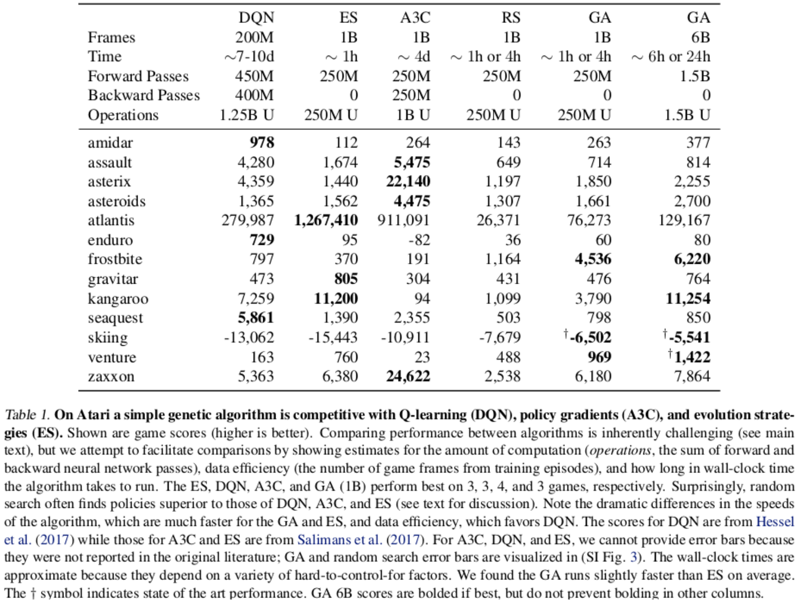
Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning

Felipe Petroski Such Vashisht Madhavan Edoardo Conti Joel Lehman Kenneth O. Stanley Jeff Clune

Uber AI Labs

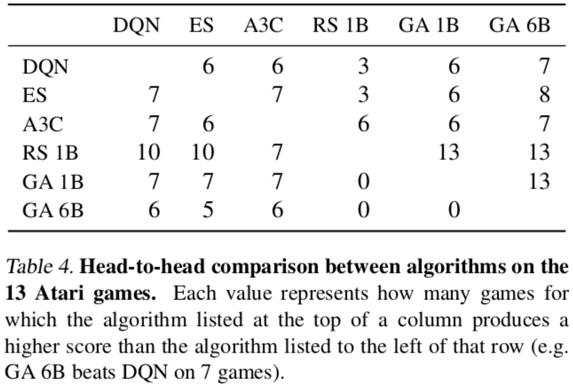
`{felipe.such, jeffclune}@uber.com`

(Do you spend your time training nets, or simulating?)



	DQN	ES	A3C	RS	GA	GA
Frames	200M	1B	1B	1B	1B	6B
Time	∼7-10d	∼ 1h	∼ 4d	∼ 1h or 4h	∼ 1h or 4h	∼ 6h or 24h
Forward Passes	450M	250M	250M	250M	250M	1.5B
Backward Passes	400M	0	250M	0	0	0
Operations	1.25B U	250M U	1B U	250M U	250M U	1.5B U
amidar	**978**	112	264	143	263	377
assault	4,280	1,674	**5,475**	649	714	814
asterix	4,359	1,440	**22,140**	1,197	1,850	2,255
asteroids	1,365	1,562	**4,475**	1,307	1,661	2,700
atlantis	279,987	**1,267,410**	911,091	26,371	76,273	129,167
enduro	**729**	95	-82	36	60	80
frostbite	797	370	191	1,164	**4,536**	**6,220**
gravitar	473	**805**	304	431	476	764
kangaroo	7,259	**11,200**	94	1,099	3,790	**11,254**
seaquest	**5,861**	1,390	2,355	503	798	850
skiing	-13,062	-15,443	-10,911	-7,679	†**-6,502**	†**-5,541**
venture	163	760	23	488	**969**	†**1,422**
zaxxon	5,363	6,380	**24,622**	2,538	6,180	7,864

Table 1. **On Atari a simple genetic algorithm is competitive with Q-learning (DQN), policy gradients (A3C), and evolution strategies (ES).** Shown are game scores (higher is better). Comparing performance between algorithms is inherently challenging (see main text), but we attempt to facilitate comparisons by showing estimates for the amount of computation (*operations*, the sum of forward and backward neural network passes), data efficiency (the number of game frames from training episodes), and how long in wall-clock time the algorithm takes to run. The ES, DQN, A3C, and GA (1B) perform best on 3, 3, 4, and 3 games, respectively. Surprisingly, random search often finds policies superior to those of DQN, A3C, and ES (see text for discussion). Note the dramatic differences in the speeds of the algorithm, which are much faster for the GA and ES, and data efficiency, which favors DQN. The scores for DQN are from Hessel et al. (2017) while those for A3C and ES are from Salimans et al. (2017). For A3C, DQN, and ES, we cannot provide error bars because they were not reported in the original literature; GA and random search error bars are visualized in (SI Fig. 3). The wall-clock times are approximate because they depend on a variety of hard-to-control-for factors. We found the GA runs slightly faster than ES on average. The † symbol indicates state of the art performance. GA 6B scores are bolded if best, but do not prevent bolding in other columns.



	DQN	ES	A3C	RS 1B	GA 1B	GA 6B
DQN		6	6	3	6	7
ES	7		7	3	6	8
A3C	7	6		6	6	7
RS 1B	10	10	7		13	13
GA 1B	7	7	7	0		13
GA 6B	6	5	6	0	0	

Table 4. **Head-to-head comparison between algorithms on the 13 Atari games.** Each value represents how many games for which the algorithm listed at the top of a column produces a higher score than the algorithm listed to the left of that row (e.g. GA 6B beats DQN on 7 games).

- Conclusion: It varies from problem to problem what is better.
  And it is suprising that “naive” black-box ES can beat elaborate RL-methods

Conclusions

- Overall, it is not so clear at all whether RL is actually better than black-box/derivative-free optimization
- For small problems where we have little function approximation or noisy gradient problems, yes; but for large scale problems?

- For discussion:
  - If you have a problem with dense rewards and smooth dynamics...?

Conclusions

- Overall, it is not so clear at all whether RL is actually better than black-box/derivative-free optimization
- For small problems where we have little function approximation or noisy gradient problems, yes; but for large scale problems?

- For discussion:
  - If you have a problem with dense rewards and smooth dynamics...?
  - If you have a sparse reward problem, but with smooth/easy dynamics...?

Conclusions

- Overall, it is not so clear at all whether RL is actually better than black-box/derivative-free optimization
- For small problems where we have little function approximation or noisy gradient problems, yes; but for large scale problems?

- For discussion:
  - If you have a problem with dense rewards and smooth dynamics...?
  - If you have a sparse reward problem, but with smooth/easy dynamics...?
  - If you have a sparse reward problem with hard dynamics...?

Conclusions

- Overall, it is not so clear at all whether RL is actually better than black-box/derivative-free optimization
- For small problems where we have little function approximation or noisy gradient problems, yes; but for large scale problems?

- For discussion:
  - If you have a problem with dense rewards and smooth dynamics...?
  - If you have a sparse reward problem, but with smooth/easy dynamics...?
  - If you have a sparse reward problem with hard dynamics...?

- RL-methods rely on reward signals/gradients that can be "propagated" through time/steps (credit assignment, Q-learning, Bellman)
- Black-box search is ignorant to this, sometimes to the better