Deep Policy Search Methods

Herke van Hoof

Deep Policy Search

In principle, all of the policy search methods discussed so far work with **any** type of policy

Like with value-based methods, the use of **deep neural networks** have become popular

What are considerations when using deep neural networks?

What are considerations when evaluating the results?

Deep Policy Search

Running example:

Deterministic policy gradients (DPG / DDPG)

Deterministic policy gradients

All policy gradients so far learned a stochastic policy

It could be beneficial to learn a deterministic policy instead

This implies the use of two policies: a stochastic policy for exploration (*behavior policy* β) next to the actor policy π

Aim: learn the actor π using data from the behaviour policy β
(π called target policy before - terminology is confusing here!)

β is typically actor + (Gaussian) noise

Deterministic policy gradients

The normal policy gradient has just one policy

$$\nabla_\theta J\left(\pi_\theta\right) = \mathbb{E}_{s\sim\mu_\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^w(s,a)\right]$$

But we only have samples of the state distribution that come from β!

Deterministic policy gradients

The normal policy gradient has just one policy

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\mu_\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^w(s,a)\right]$$

But we only have samples of the state distribution that come from β!

We'll optimise a different objective

$$J_\beta(\pi_\theta) = \int_{\mathcal{S}} \mu^\beta(s) V^\pi(s) \mathrm{d}s$$

$$= \int_{\mathcal{S}} \mu^\beta(s) Q^\pi(s, \pi_\theta(s))\, \mathrm{d}s$$

(Note: only integral over state space, not actions!)

Deterministic policy gradients

Note that the objective looks like the one from Q-learning!

$$J_\beta\left(\pi_\theta\right) = \int_{\mathcal{S}} \mu^\beta(s) V^\pi(s) \mathrm{d}s$$

$$= \int_{S} \mu^\beta(s) Q^\pi\left(s, \pi_\theta(s)\right) \mathrm{d}s$$

The greedy policy maximises this objective (but we cannot obtain it directly in continuous action spaces)

Note that like in Q-learning, we do not need importance weights, as the 'target' doesn't depend on sampled actions.

Deterministic policy gradients

The off-policy deterministic policy gradient is

$$\nabla_\theta J_\beta(\pi_\theta) = \int_{\mathcal{S}} \mu^\beta(s) \nabla_\theta Q^\pi(s, \pi_\theta(s)) \mathrm{d}s$$

$$= \mathbb{E}_{s\sim\mu^\beta}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s,a)\big|_{a=\pi_\theta(s)}\right]$$

β only in sampling distribution

Deterministic policy gradients

The off-policy deterministic policy gradient is

$$\nabla_\theta J_\beta(\pi_\theta) = \int_{\mathcal{S}} \mu^\beta(s) \nabla_\theta Q^\pi(s, \pi_\theta(s)) \mathrm{d}s$$

$$= \mathbb{E}_{s\sim\mu^\beta}\left[\nabla_\theta \pi_\theta(s) \nabla_a \boxed{Q^\pi}(s,a)|_{a=\pi_\theta(s)}\right]$$

β only in sampling distribution

leading to the DPG algorithm with Q learning update

$$\delta_t = r_t + \gamma \boxed{Q^w}(s_{t+1}, \boxed{\pi_\theta}(s_{t+1})) - Q^w(s_t, a_t)$$ TD-errors

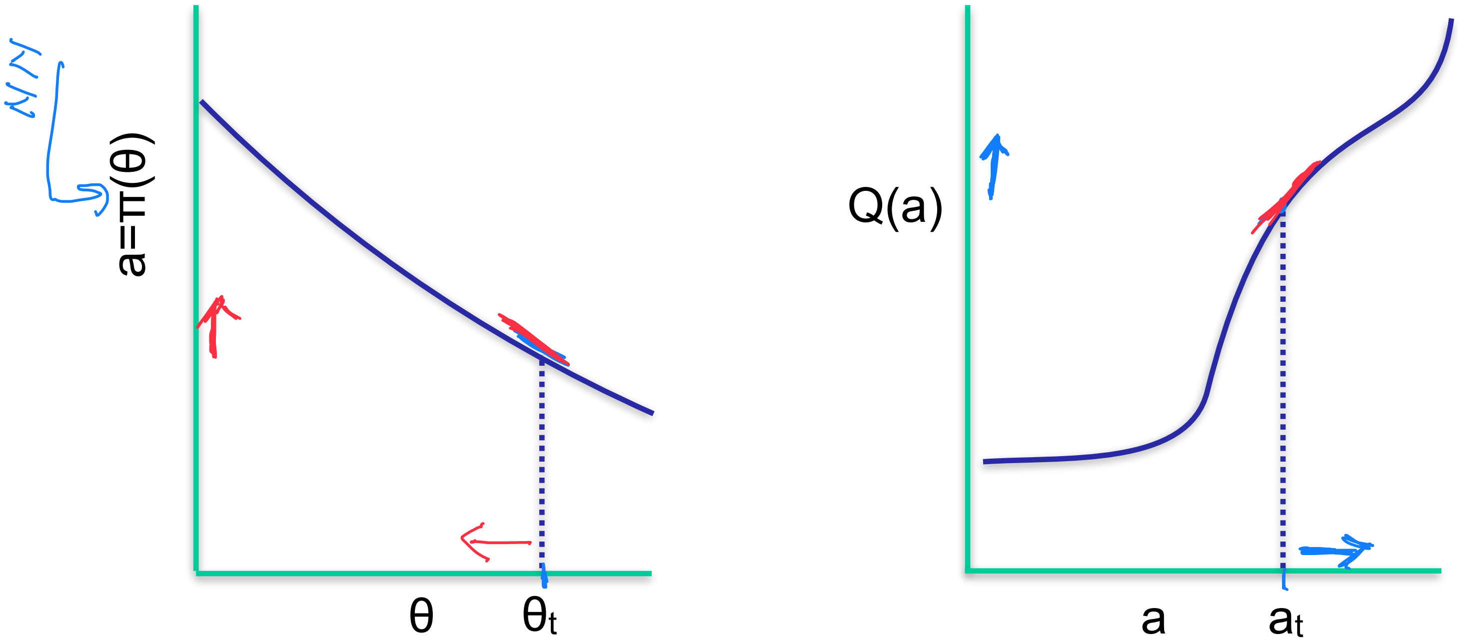
$$\boxed{w_{t+1}} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)$$ Q-update

semi-gradient SARSA

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \pi_\theta(s_t) \nabla_a Q^w(s_t, a_t)|_{a=\pi_\theta(s)}$$ Policy update, doesn't directly depend on β!

Deterministic policy gradient

So how can we imagine this for continuous actions?



Of course, the functions π and Q need to be differentiable!

Deterministic policy gradient

28min

We only have deterministic policy gradients for continuous actions!

Parameters.

For discrete, deterministic actions, a tiny change in weights will

- not change the expected return at all (gradient = 0)
- *or* cause a jump in the expected return (gradient undefined)

Note that for discrete, stochastic actions, tiny weight changes tend to cause small differences in probabilities, thus small changes to expected returns…

- so stochastic gradients are ok for discrete actions

Deterministic policy gradient

lec 7,8,9
reinforce, actor critic, gpomdp

lec 10

	Discrete actions	Continuous actions
Stochastic policy gradients	✓	✓
Deterministic policy gradients	✗ (no gradients $\nabla_\theta \mu_\theta(s), \nabla_a Q^\mu(s,a)$)	✓
Critic-only methods	✓	✗ (how to extract policy?)

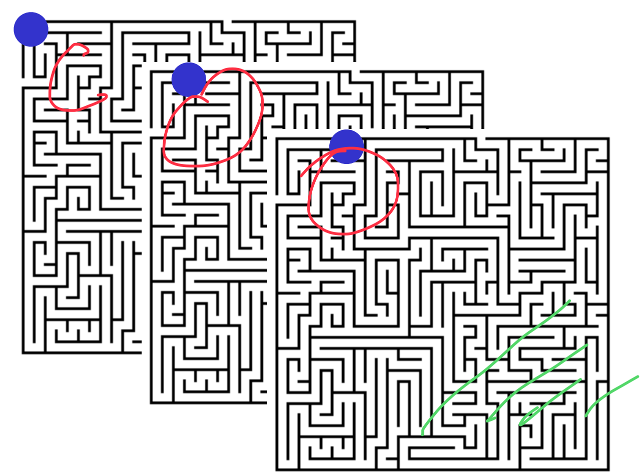
Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

Problem 1: Subsequent samples tend to be highly correlated

Problem 2: Data is only used once

Solution ?



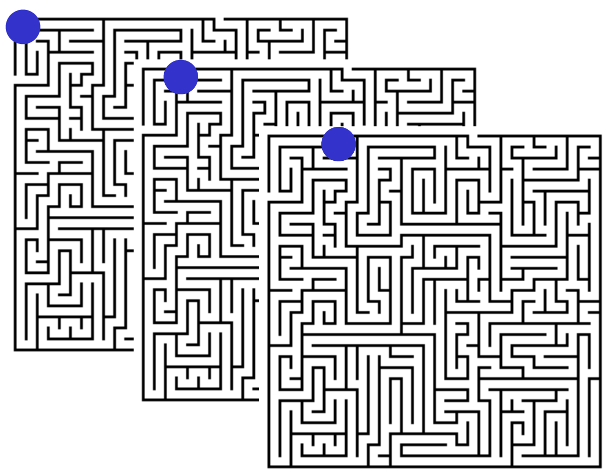
Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

Problem 1: Subsequent samples tend to be highly correlated

Problem 2: Data is only used once

Solution: add experience replay



Experience replay changes data distribution β
→*but update doesn't depend on* β*!*
(what is β *now?)*

Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

Problem 3: TD-learning is unstable way to learn critic

Network being updated is also used to calculate target value: can lead to divergence

Deep DPG

In DQN, divergence addressed using “target-networks” that are kept fixed for number of timesteps, then copied from learned Q

Here, target Q-network slowly tracks the learned Q, and an additional target policy is introduced tracking the actor policy

terminology from DDPG paper:
behaviour policy: generating the data. Actor + noise
actor policy: what we are learning, called ‘target’ before
target policy: used to select action for Q-update,
analogous to ‘target Q-network’

Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

Problem 4: Features often not on same scale

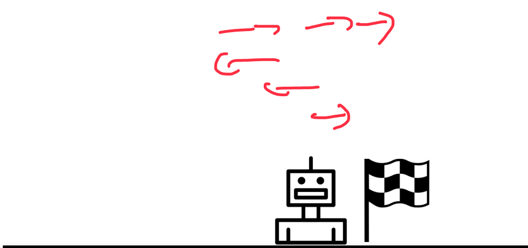
Similar to what we discussed in natural gradients, different scales can distort the learning process

Popular solution from deep learning: Batch normalisation (normalizing the input within each mini-batch at **each layer**)

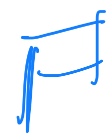
Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

Problem 5: Independent noise doesn't explore well



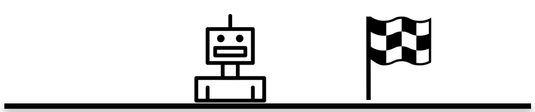
position



Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

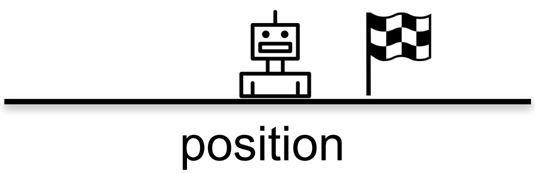
Problem 5: Independent noise doesn't explore well



Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

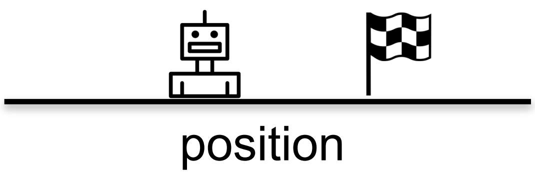
Problem 5: Independent noise doesn't explore well



Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

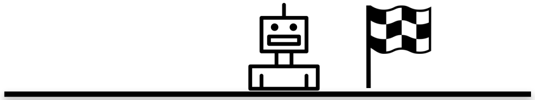
Problem 5: Independent noise doesn't explore well



Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

Problem 5: Independent noise doesn't explore well

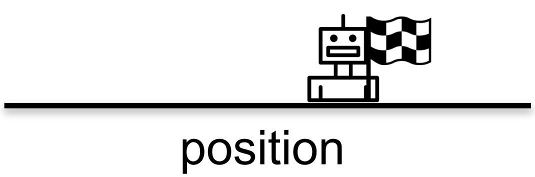


position

Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

Problem 5: Independent noise doesn’t explore well



Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

Problem 5: Independent noise doesn't explore well

Deep DPG

Deep DPG = DPG + modification to use neural nets to generalise

Problem 5: Independent noise doesn't explore well

The paper adds **correlated exploration**: More likely to continue in the same random direction as last time step

Of course, this changes β, but again that does not matter…

Deep DPG

$p(a \mid s, a_{1:t-1}, a_{t:T-1})$

Deep DPG = DPG + modification to use neural nets to generalise

Re-use samples and make samples less correlated
→ *add experience replay* * *

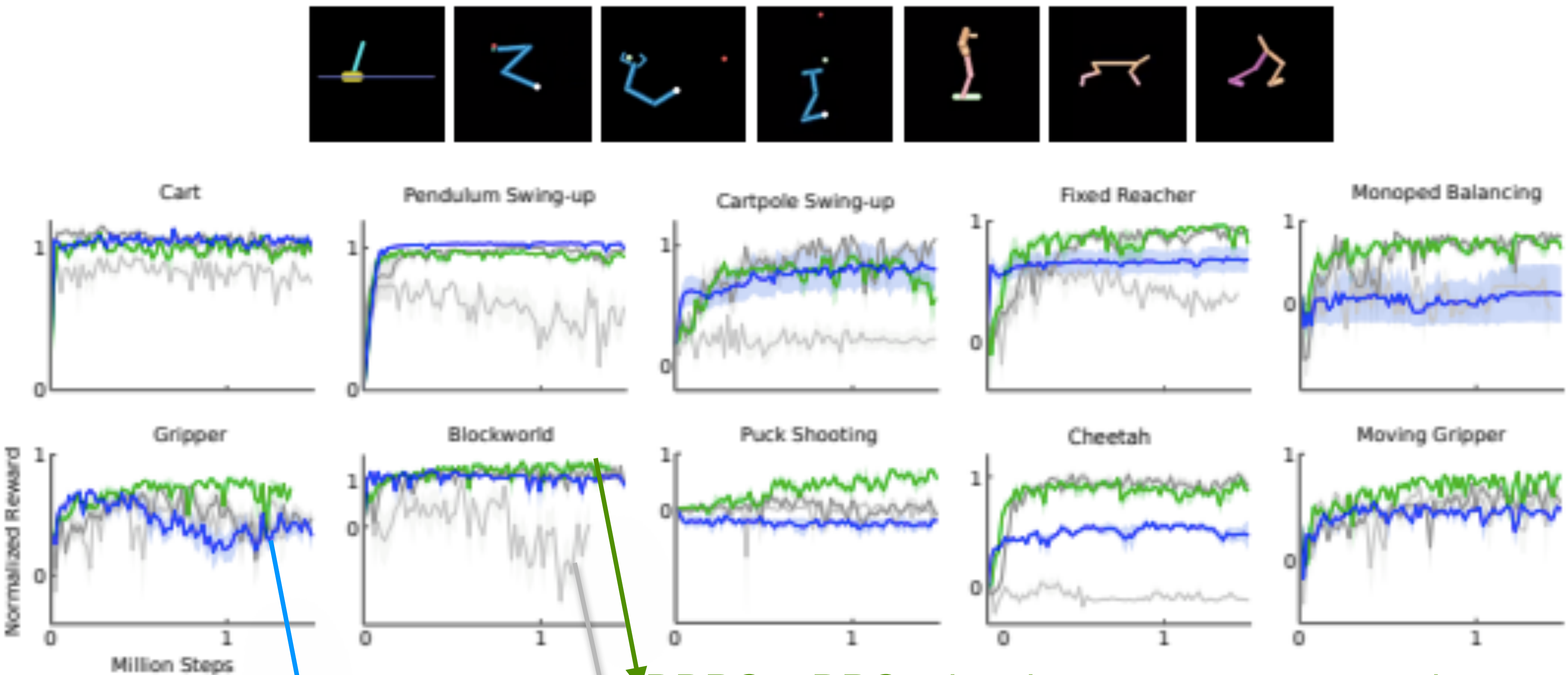
Q learning unstable
→ *add target networks for actor μ and Q-network* *

Features often not in same scale
→*use batch normalisation*

Independent noise doesn't explore well
→*use correlated noise* *

*Use: Update doesn't depend on β, use any source of samples
*Similar to DQN

Results

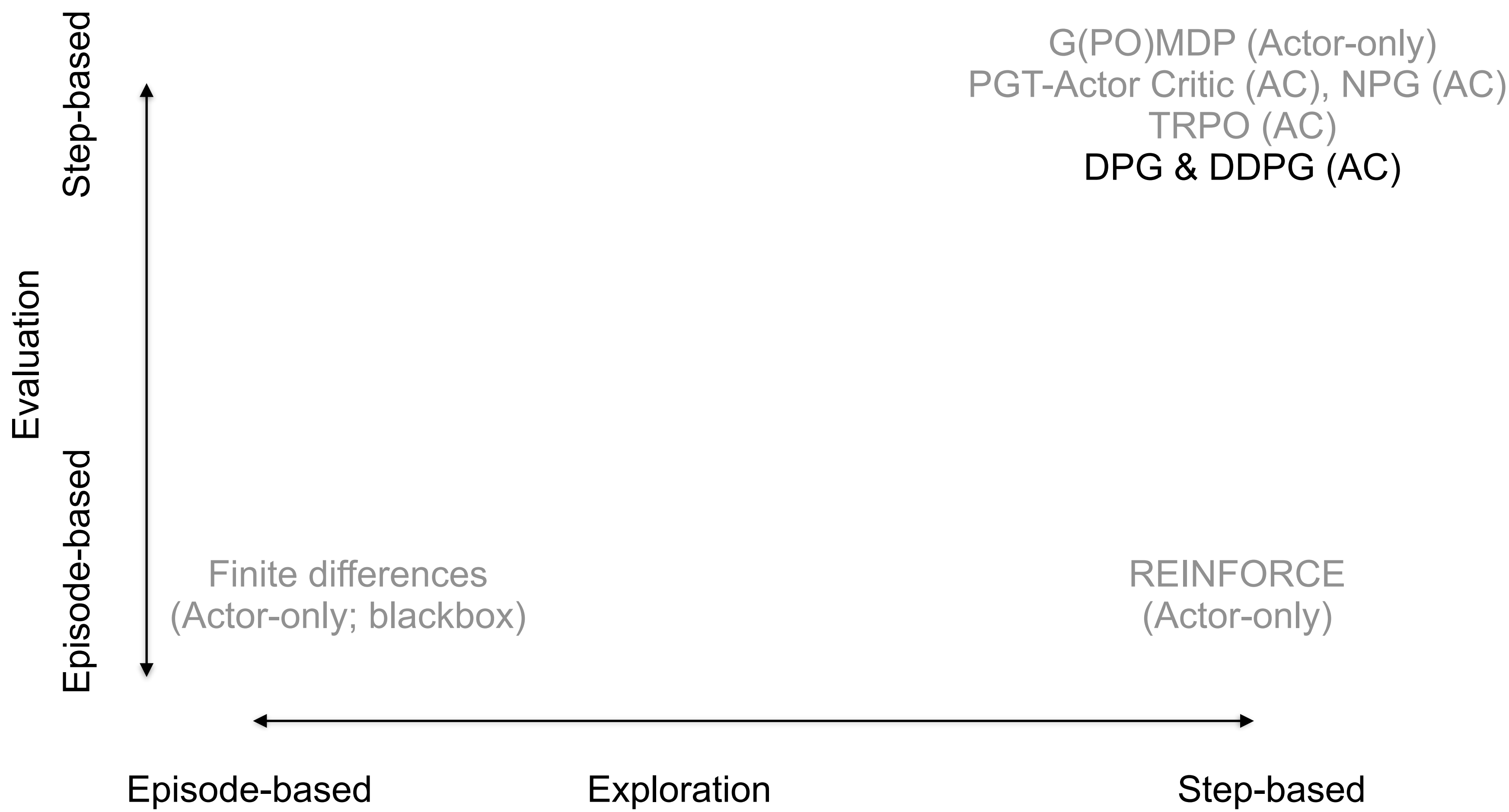


DDPG = DPG + batchnorm + target networks

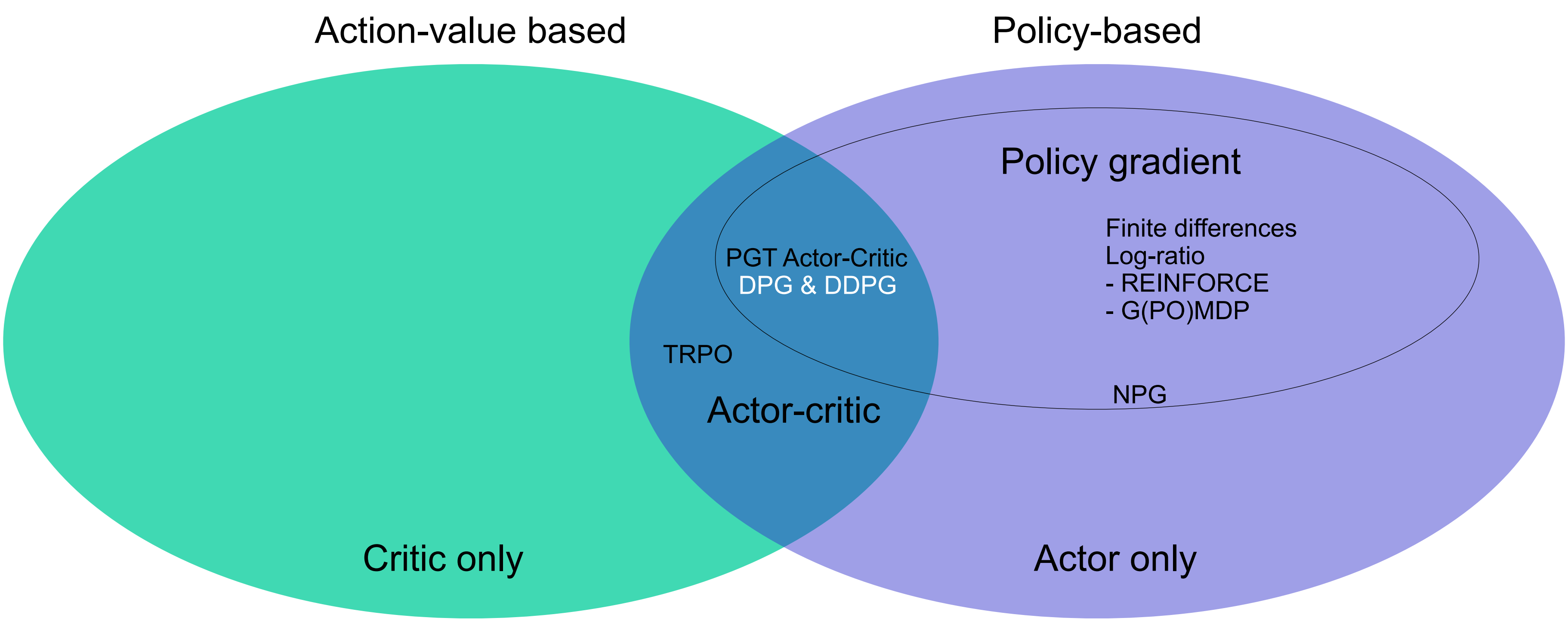
from images

DPG + batchnorm

Comparison


G(PO)MDP (Actor-only)
PGT-Actor Critic (AC), NPG (AC),
TRPO (AC)
DPG & DDPG (AC)
Step-based
Evaluation
Episode-based
Finite differences
(Actor-only; blackbox)
REINFORCE
(Actor-only)
Episode-based
Exploration
Step-based

Policies and action-values



Deep RL algorithms

Similarly, there are many other proposed algorithms that try to resolve certain problems in training deep neural networks

- A3C: Actor critic method that uses experience from multiple parallel threads to make data more independent
- ACER: Combine off-policy estimation of Q^{π} and off-policy estimation of policy gradient to allow "A3C with experience replay"
- PPO: Approximate TRPO in a way that doesn't require 2nd order statistic and allows dropout and parameter sharing
- SAC: Find policies that explore yet get close to maximimal reward by learning a policy that looks like the softmax of Q
- TD3: Improve DDPG to avoid maximisation bias, using tricks like 'double learning'

Deep RL in practice

Which algorithm to choose? Is one ‘best’?

Debugging RL algorithms

Comparison

		Lecture 7	Lecture 9			Lecture 9			This lecture
Task	Random	REINFORCE	TNPG	RWR	REPS	TRPO	CEM	CMA-ES	DDPG
Cart-Pole Balancing	77.1 ± 0.0	4693.7 ± 14.0	**3986.4 ± 748.9**	**4861.5 ± 12.3**	565.6 ± 137.6	**4869.8 ± 37.6**	4815.4 ± 4.8	2440.4 ± 568.3	4634.4 ± 87.8
Inverted Pendulum*	−153.4 ± 0.2	13.4 ± 18.0	**209.7 ± 55.5**	84.7 ± 13.8	−113.3 ± 4.6	**247.2 ± 76.1**	38.2 ± 25.7	−40.1 ± 5.7	40.0 ± 244.6
Mountain Car	−415.4 ± 0.0	−67.1 ± 1.0	**-66.5 ± 4.5**	−79.4 ± 1.1	−275.6 ± 166.3	**-61.7 ± 0.9**	−66.0 ± 2.4	−85.0 ± 7.7	−288.4 ± 170.3
Acrobot	−1904.5 ± 1.0	−508.1 ± 91.0	−395.8 ± 121.2	−352.7 ± 35.9	−1001.5 ± 10.8	−326.0 ± 24.4	−436.8 ± 14.7	−785.6 ± 13.1	**-223.6 ± 5.8**
Double Inverted Pendulum*	149.7 ± 0.1	4116.5 ± 65.2	**4455.4 ± 37.6**	3614.8 ± 368.1	446.7 ± 114.8	**4412.4 ± 50.4**	2566.2 ± 178.9	1576.1 ± 51.3	2863.4 ± 154.0
Swimmer*	−1.7 ± 0.1	92.3 ± 0.1	**96.0 ± 0.2**	60.7 ± 5.5	3.8 ± 3.3	**96.0 ± 0.2**	68.8 ± 2.4	64.9 ± 1.4	85.8 ± 1.8
Hopper	8.4 ± 0.0	714.0 ± 29.3	**1155.1 ± 57.9**	553.2 ± 71.0	86.7 ± 17.6	**1183.3 ± 150.0**	63.1 ± 7.8	20.3 ± 14.3	267.1 ± 43.5
2D Walker	−1.7 ± 0.0	506.5 ± 78.8	**1382.6 ± 108.2**	136.0 ± 15.9	−37.0 ± 38.1	**1353.8 ± 85.0**	84.5 ± 19.2	77.1 ± 24.3	318.4 ± 181.6
Half-Cheetah	−90.8 ± 0.3	1183.1 ± 69.2	**1729.5 ± 184.6**	376.1 ± 28.2	34.5 ± 38.0	**1914.0 ± 120.1**	330.4 ± 274.8	441.3 ± 107.6	**2148.6 ± 702.7**
Ant*	13.4 ± 0.7	548.3 ± 55.5	**706.0 ± 127.7**	37.6 ± 3.1	39.0 ± 9.8	**730.2 ± 61.3**	49.2 ± 5.9	17.8 ± 15.5	326.2 ± 20.8
Simple Humanoid	41.5 ± 0.2	128.1 ± 34.0	**255.0 ± 24.5**	93.3 ± 17.4	28.3 ± 4.7	**269.7 ± 40.3**	60.6 ± 12.9	28.7 ± 3.9	99.4 ± 28.1
Full Humanoid	13.2 ± 0.1	262.2 ± 10.5	**288.4 ± 25.2**	46.7 ± 5.6	41.7 ± 6.1	**287.0 ± 23.4**	36.9 ± 2.9	N/A ± N/A	119.0 ± 31.2
Cart-Pole Balancing (LS)*	77.1 ± 0.0	420.9 ± 265.5	**945.1 ± 27.8**	68.9 ± 1.5	898.1 ± 22.1	**960.2 ± 46.0**	227.0 ± 223.0	68.0 ± 1.6	
Inverted Pendulum (LS)	−122.1 ± 0.1	−13.4 ± 3.2	**0.7 ± 6.1**	−107.4 ± 0.2	−87.2 ± 8.0	**4.5 ± 4.1**	−81.2 ± 33.2	−62.4 ± 3.4	
Mountain Car (LS)	−83.0 ± 0.0	−81.2 ± 0.6	**-65.7** ± 9.0	−81.7 ± 0.1	−82.6 ± 0.4	**-64.2 ± 9.5**	**-68.9 ± 1.3**	**-73.2 ± 0.6**	
Acrobot (LS)*	−393.2 ± 0.0	−128.9 ± 11.6	**-84.6 ± 2.9**	−235.9 ± 5.3	−379.5 ± 1.4	**-83.3 ± 9.9**	−149.5 ± 15.3	−159.9 ± 7.5	
Cart-Pole Balancing (NO)*	101.4 ± 0.1	616.0 ± 210.8	**916.3 ± 23.0**	93.8 ± 1.2	99.6 ± 7.2	606.2 ± 122.2	181.4 ± 32.1	104.4 ± 16.0	
Inverted Pendulum (NO)	−122.2 ± 0.1	6.5 ± 1.1	**11.5 ± 0.5**	−110.0 ± 1.4	−119.3 ± 4.2	**10.4 ± 2.2**	−55.6 ± 16.7	−80.3 ± 2.8	
Mountain Car (NO)	−83.0 ± 0.0	−74.7 ± 7.8	**-64.5 ± 8.6**	−81.7 ± 0.1	−82.9 ± 0.1	**-60.2 ± 2.0**	−67.4 ± 1.4	−73.5 ± 0.5	
Acrobot (NO)*	−393.5 ± 0.0	**-186.7 ± 31.3**	**-164.5 ± 13.4**	−233.1 ± 0.4	−258.5 ± 14.0	**-149.6 ± 8.6**	−213.4 ± 6.3	−236.6 ± 6.2	
Cart-Pole Balancing (SI)*	76.3 ± 0.1	431.7 ± 274.1	**980.5 ± 7.3**	69.0 ± 2.8	702.4 ± 196.4	**980.3 ± 5.1**	746.6 ± 93.2	71.6 ± 2.9	
Inverted Pendulum (SI)	−121.8 ± 0.2	−5.3 ± 5.6	**14.8 ± 1.7**	−108.7 ± 4.7	−92.8 ± 23.9	**14.1 ± 0.9**	−51.8 ± 10.6	−63.1 ± 4.8	
Mountain Car (SI)	−82.7 ± 0.0	−63.9 ± 0.2	**-61.8 ± 0.4**	−81.4 ± 0.1	−80.7 ± 2.3	**-61.6 ± 0.4**	−63.9 ± 1.0	−66.9 ± 0.6	
Acrobot (SI)*	−387.8 ± 1.0	**-169.1 ± 32.3**	**-156.6 ± 38.9**	−233.2 ± 2.6	−216.1 ± 7.7	**-170.9 ± 40.3**	−250.2 ± 13.7	−245.0 ± 5.5	
Swimmer + Gathering	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Ant + Gathering	−5.8 ± 5.0	−0.1 ± 0.1	−0.4 ± 0.1	−5.5 ± 0.5	−6.7 ± 0.7	−0.4 ± 0.0	−4.7 ± 0.7	N/A ± N/A	−0.3 ± 0.3
Swimmer + Maze	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Ant + Maze	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	N/A ± N/A	0.0 ± 0.0

Duan et al., Benchmarking Deep Reinforcement Learning for continuous control, 2016

Comparison

Task	Random	REINFORCE	TNPG	RWR	REPS	TRPO	CEM	CMA-ES	DDPG
		Lecture 7	Lecture 9			Lecture 9			This lecture
Cart-Pole Balancing	77.1 ± 0.0	4693.7 ± 14.0	**3986.4 ± 748.9**	**4861.5 ± 12.3**	565.6 ± 137.6	**4869.8 ± 37.6**	4815.4 ± 4.8	2440.4 ± 568.3	4634.4 ± 87.8
Inverted Pendulum*	−153.4 ± 0.2	13.4 ± 18.0	**209.7 ± 55.5**	84.7 ± 13.8	−113.3 ± 4.6	**247.2 ± 76.1**	38.2 ± 25.7	−40.1 ± 5.7	40.0 ± 244.6
Mountain Car	−415.4 ± 0.0	−67.1 ± 1.0	**-66.5 ± 4.5**	−79.4 ± 1.1	−275.6 ± 166.3	**-61.7 ± 0.9**	−66.0 ± 2.4	−85.0 ± 7.7	−288.4 ± 170.3
Acrobot	−1904.5 ± 1.0	−508.1 ± 91.0	−395.8 ± 121.2	−352.7 ± 35.9	−1001.5 ± 10.8	−326.0 ± 24.4	−436.8 ± 14.7	−785.6 ± 13.1	**-223.6 ± 5.8**
Double Inverted Pendulum*	149.7 ± 0.1	4116.5 ± 65.2	**4455.4 ± 37.6**	3614.8 ± 368.1	446.7 ± 114.8	**4412.4 ± 50.4**	2566.2 ± 178.9	1576.1 ± 51.3	2863.4 ± 154.0
Swimmer*	−1.7 ± 0.1	92.3 ± 0.1	**96.0 ± 0.2**	60.7 ± 5.5	3.8 ± 3.3	**96.0 ± 0.2**	68.8 ± 2.4	64.9 ± 1.4	85.8 ± 1.8
Hopper	8.4 ± 0.0	714.0 ± 29.3	**1155.1 ± 57.9**	553.2 ± 71.0	86.7 ± 17.6	**1183.3 ± 150.0**	63.1 ± 7.8	20.3 ± 14.3	267.1 ± 43.5
2D Walker	−1.7 ± 0.0	506.5 ± 78.8	**1382.6 ± 108.2**	136.0 ± 15.9	−37.0 ± 38.1	**1353.8 ± 85.0**	84.5 ± 19.2	77.1 ± 24.3	318.4 ± 181.6
Half-Cheetah	−90.8 ± 0.3	1183.1 ± 69.2	**1729.5 ± 184.6**	376.1 ± 28.2	34.5 ± 38.0	**1914.0 ± 120.1**	330.4 ± 274.8	441.3 ± 107.6	**2148.6 ± 702.7**
Ant*	13.4 ± 0.7	548.3 ± 55.5	**706.0 ± 127.7**	37.6 ± 3.1	39.0 ± 9.8	**730.2 ± 61.3**	49.2 ± 5.9	17.8 ± 15.5	326.2 ± 20.8
Simple Humanoid	41.5 ± 0.2	128.1 ± 34.0	**255.0 ± 24.5**	93.3 ± 17.4	28.3 ± 4.7	**269.7 ± 40.3**	60.6 ± 12.9	28.7 ± 3.9	99.4 ± 28.1
Full Humanoid	13.2 ± 0.1	262.2 ± 10.5	**288.4 ± 25.2**	46.7 ± 5.6	41.7 ± 6.1	**287.0 ± 23.4**	36.9 ± 2.9	N/A ± N/A	119.0 ± 31.2
Cart-Pole Balancing (LS)*	77.1 ± 0.0	420.9 ± 265.5	**945.1 ± 27.8**	68.9 ± 1.5	898.1 ± 22.1	**960.2 ± 46.0**	227.0 ± 223.0	68.0 ± 1.6	
Inverted Pendulum (LS)	−122.1 ± 0.1	−13.4 ± 3.2	**0.7 ± 6.1**	−107.4 ± 0.2	−87.2 ± 8.0	**4.5 ± 4.1**	−81.2 ± 33.2	−62.4 ± 3.4	
Mountain Car (LS)	−83.0 ± 0.0	−81.2 ± 0.6	**-65.7** ± 9.0	−81.7 ± 0.1	−82.6 ± 0.4	**-64.2 ± 9.5**	**-68.9 ± 1.3**	**-73.2 ± 0.6**	
Acrobot (LS)*	−393.2 ± 0.0	−128.9 ± 11.6	**-84.6 ± 2.9**	−235.9 ± 5.3	−379.5 ± 1.4	**-83.3 ± 9.9**	−149.5 ± 15.3	−159.9 ± 7.5	
Cart-Pole Balancing (NO)*	101.4 ± 0.1	616.0 ± 210.8	**916.3 ± 23.0**	93.8 ± 1.2	99.6 ± 7.2	606.2 ± 122.2	181.4 ± 32.1	104.4 ± 16.0	
Inverted Pendulum (NO)	−122.2 ± 0.1	6.5 ± 1.1	**11.5 ± 0.5**	−110.0 ± 1.4	−119.3 ± 4.2	**10.4 ± 2.2**	−55.6 ± 16.7	−80.3 ± 2.8	
Mountain Car (NO)	−83.0 ± 0.0	−74.7 ± 7.8	**-64.5 ± 8.6**	−81.7 ± 0.1	−82.9 ± 0.1	**-60.2 ± 2.0**	−67.4 ± 1.4	−73.5 ± 0.5	
Acrobot (NO)*	−393.5 ± 0.0	**-186.7 ± 31.3**	**-164.5 ± 13.4**	−233.1 ± 0.4	−258.5 ± 14.0	**-149.6 ± 8.6**	−213.4 ± 6.3	−236.6 ± 6.2	
Cart-Pole Balancing (SI)*	76.3 ± 0.1	431.7 ± 274.1	**980.5 ± 7.3**	69.0 ± 2.8	702.4 ± 196.4	**980.3 ± 5.1**	746.6 ± 93.2	71.6 ± 2.9	
Inverted Pendulum (SI)	−121.8 ± 0.2	−5.3 ± 5.6	**14.8 ± 1.7**	−108.7 ± 4.7	−92.8 ± 23.9	**14.1 ± 0.9**	−51.8 ± 10.6	−63.1 ± 4.8	
Mountain Car (SI)	−82.7 ± 0.0	−63.9 ± 0.2	**-61.8 ± 0.4**	−81.4 ± 0.1	−80.7 ± 2.3	**-61.6 ± 0.4**	−63.9 ± 1.0	−66.9 ± 0.6	
Acrobot (SI)*	−387.8 ± 1.0	**-169.1 ± 32.3**	**-156.6 ± 38.9**	−233.2 ± 2.6	−216.1 ± 7.7	**-170.9 ± 40.3**	−250.2 ± 13.7	−245.0 ± 5.5	
Swimmer + Gathering	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Ant + Gathering	−5.8 ± 5.0	−0.1 ± 0.1	−0.4 ± 0.1	−5.5 ± 0.5	−6.7 ± 0.7	−0.4 ± 0.0	−4.7 ± 0.7	N/A ± N/A	−0.3 ± 0.3
Swimmer + Maze	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Ant + Maze	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	N/A ± N/A	0.0 ± 0.0

Duan et al., Benchmarking Deep Reinforcement Learning for continuous control, 2016

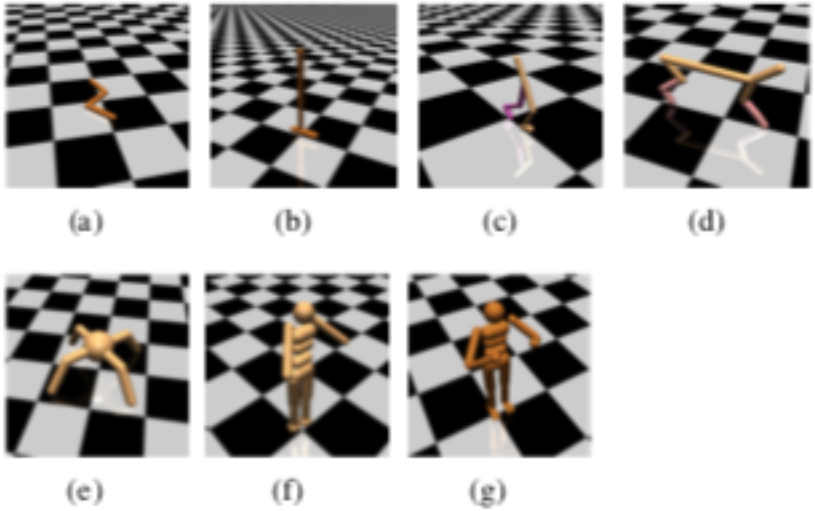
Comparison

On which task?

Which implementation?

What to measure?

How to tune?



Duan et al., Benchmarking Deep Reinforcement Learning for continuous control, 2016

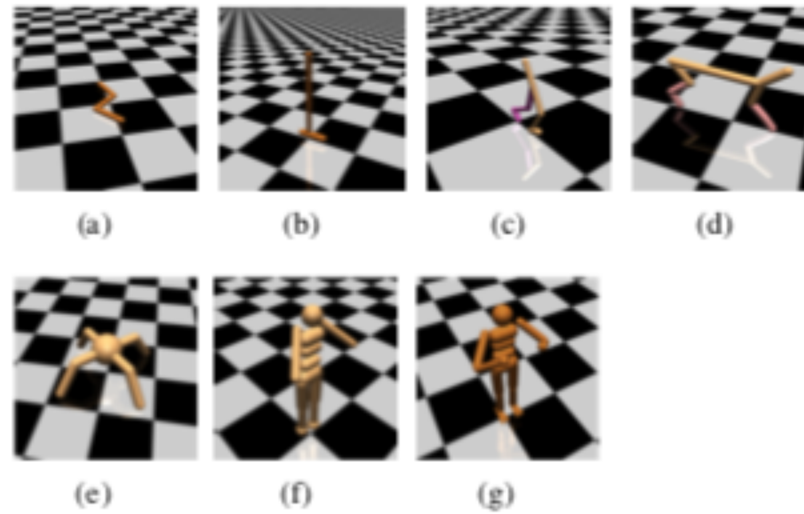
Comparison

On which task?

Which implementation?

What to measure?

How to tune?



Duan et al., Benchmarking Deep Reinforcement Learning for continuous control, 2016

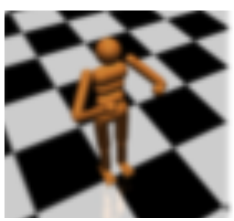
These issues are good to keep in mind when you do your own experiments (lab!) but also when you are reading a paper!

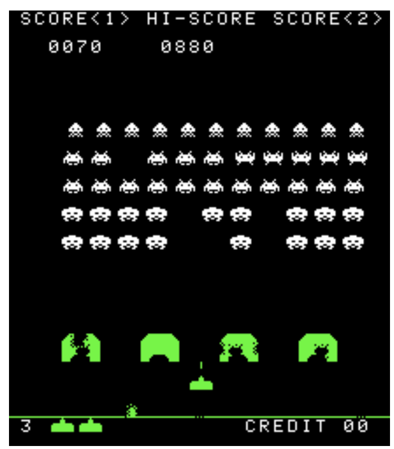
On which task?

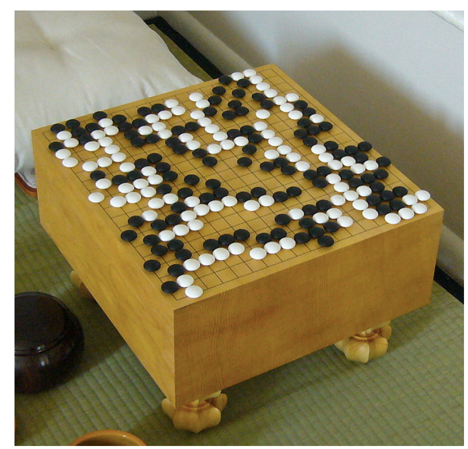
Start from the research question and methods you have

Which environments are suitable to answer the question?

- Continuous control tasks lend themselves to actor critic methods
- Pixel-based task can show whether complex input data can be handled
- Highly complex tasks show whether a method scales with having lots of compute and training data available







On which task?

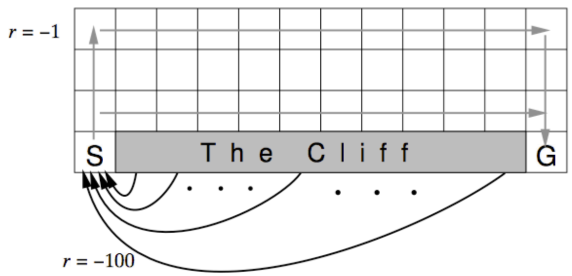
Start from the research question and methods you have

Which environments are suitable to answer the question?

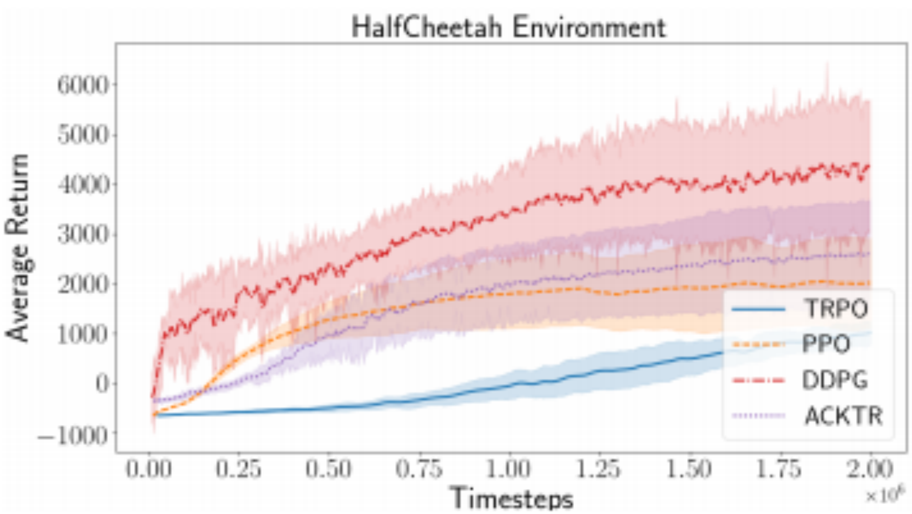
- For many specific question, we can look at specific environments:

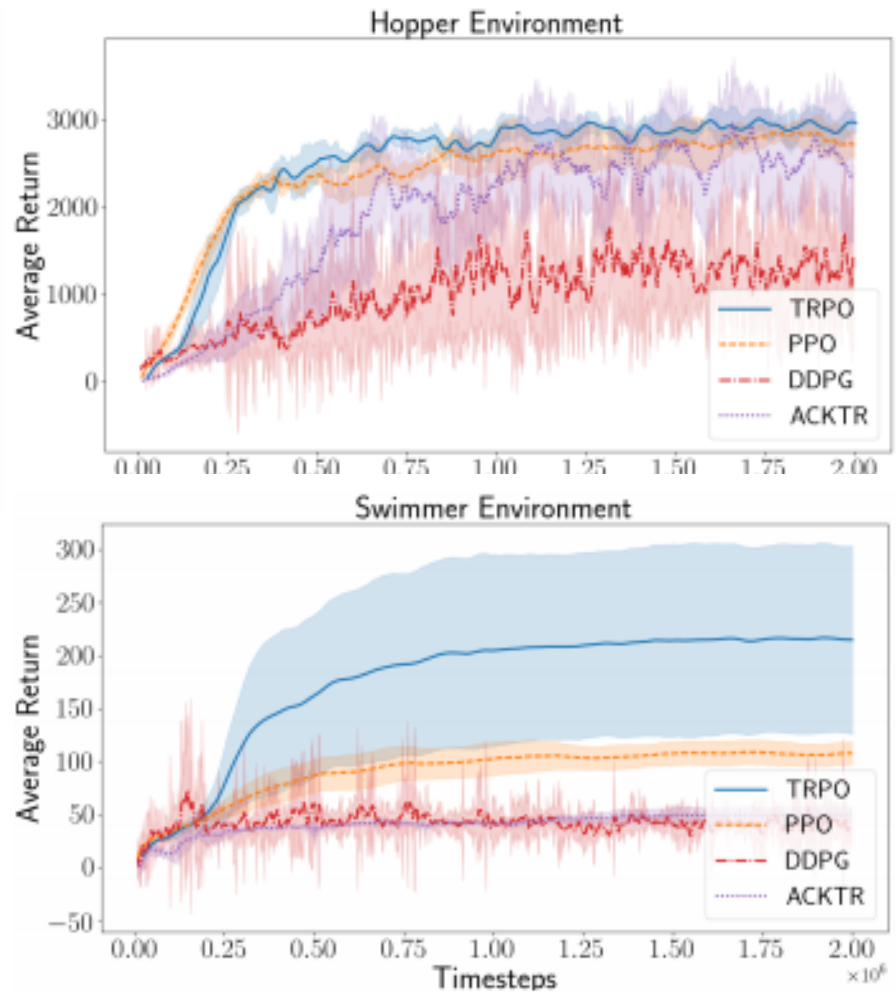
    E.g. cliff world used to show difference on-policy and off-policy

- It is often good to combine an experiment on a specific ‘toy’ environment with a more ‘realistic' experiment



On which task?





How reliable is result on one environment?

Henderson et al., Deep Reinforcement Learning that Matters, 2017

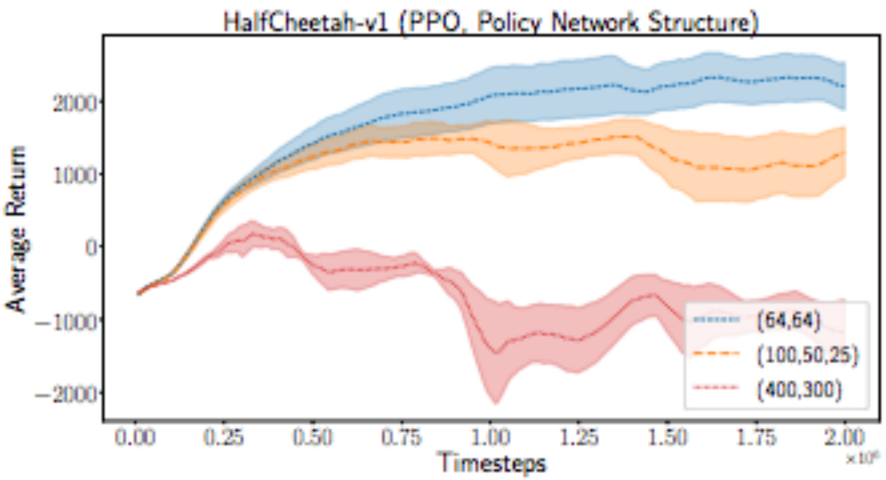
Parameters & Architecture

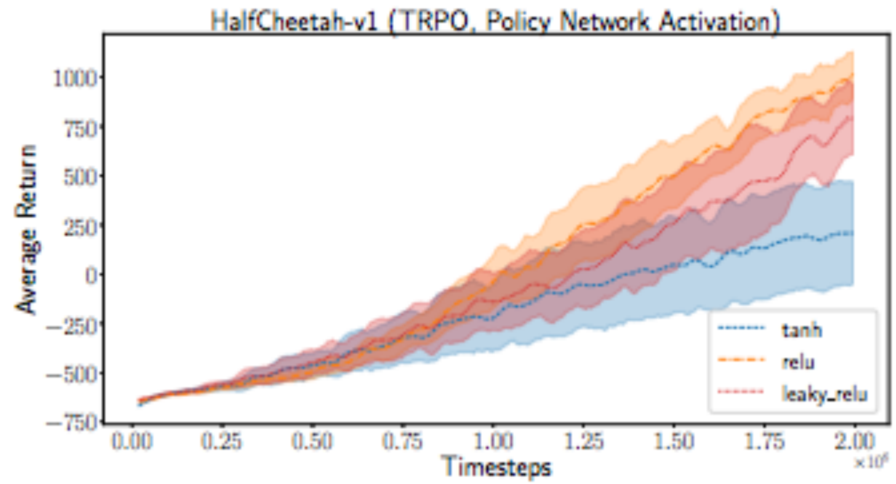
Deep reinforcement learning can be very sensitive to parameters, architecture, implementation details and even the random seed

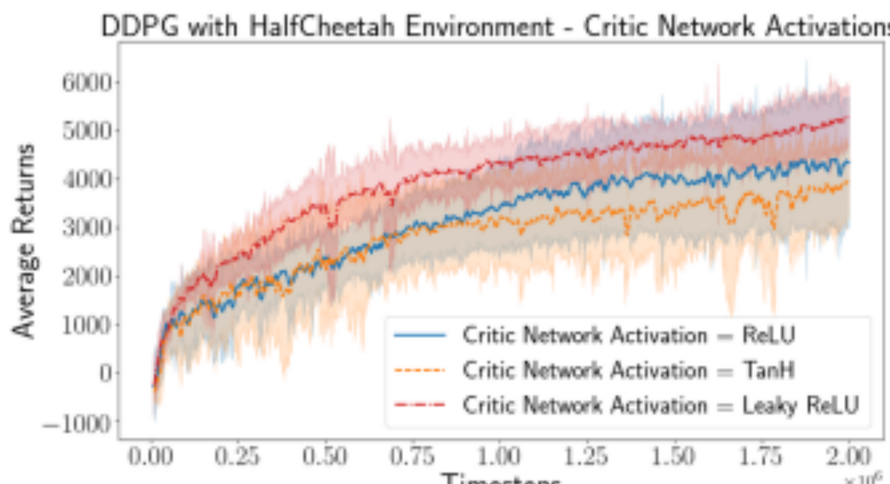
This has implications for what to report, how to set-up experiments, and how to interpret a paper

Some examples from "Deep reinforcement learning that matters" [Henderson et al., 2017]

Network structure

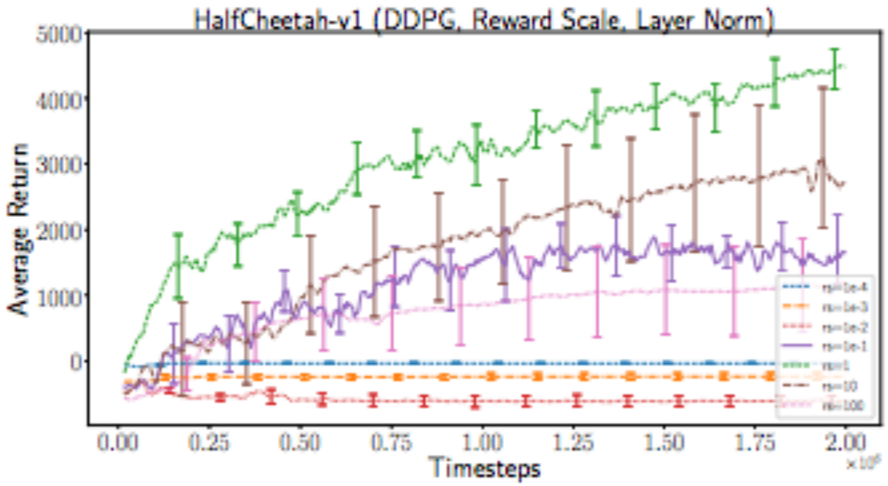


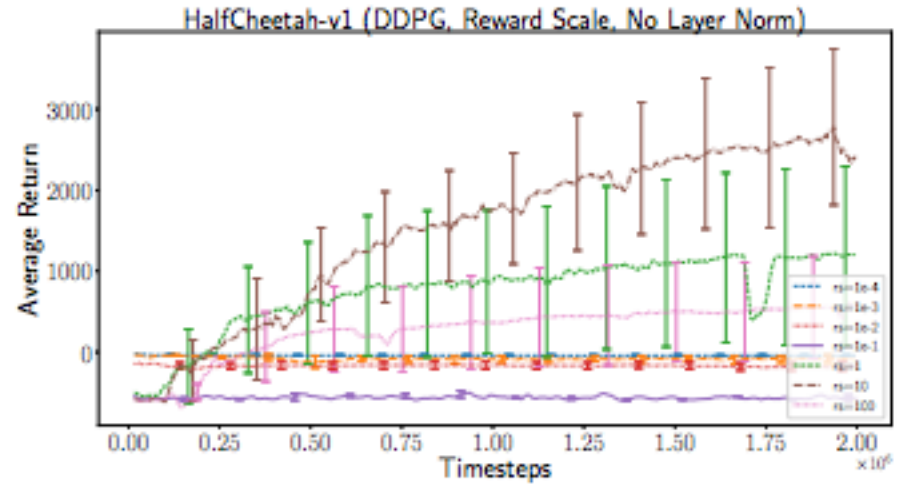




Henderson et al., Deep Reinforcement Learning that Matters, 2017

Reward scaling and layer normalisation



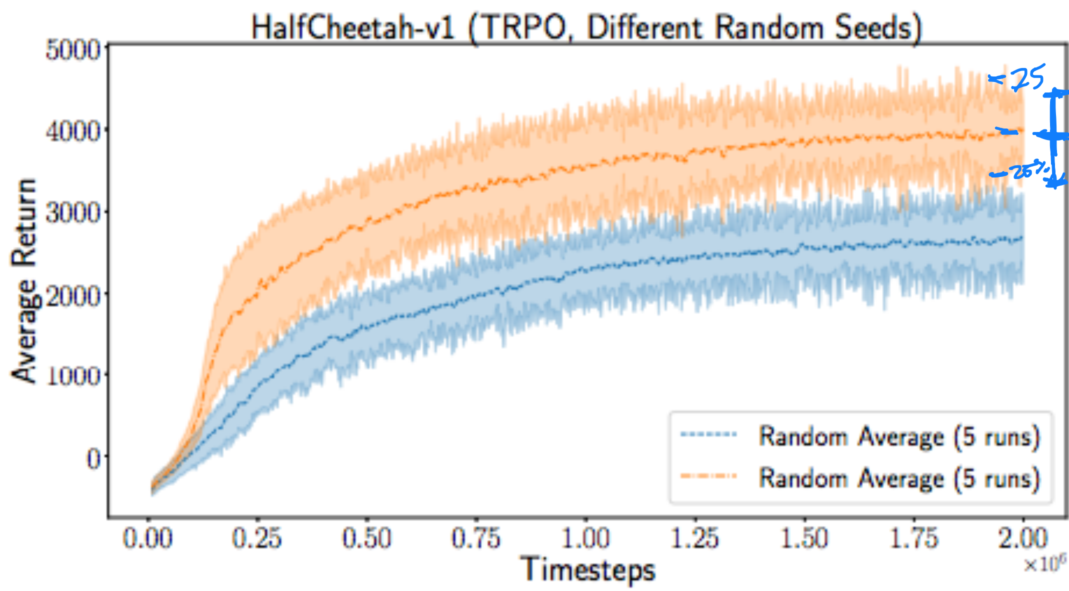


What does this mean for comparison experiments?

What are the consequences of this for reporting?

Henderson et al., Deep Reinforcement Learning that Matters, 2017

Random seeds



std dev, sterr.

mean median

How many random seeds is enough?

What is reported (mean, max, median)?

Henderson et al., Deep Reinforcement Learning that Matters, 2017

Random seeds

Adding a hyperparameter and optimising will **never** decrease performance (worst case, set it to old default)

This sounds like **overfitting**

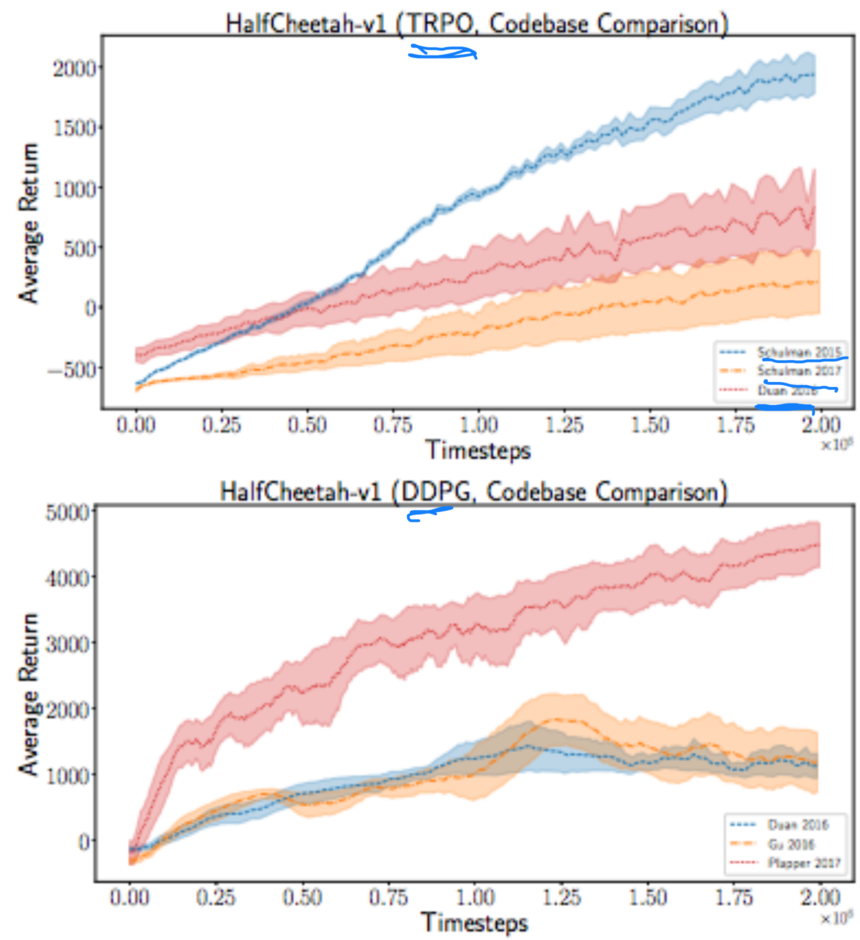
Are hyperparameters tuned specifically to certain set of random seeds?

- Tune the hyperparameters on a certain (set of) random seeds
- Report evaluation of independently trained models on a different set of random seeds. Use sufficient random seeds!
- Clearly, choosing the ‘best random seed’ is a text book example of overfitting

Codebase / implementation

What does this mean for reporting

- when a new algorithm is introduced?
- when using another method as baseline?



Henderson et al., Deep Reinforcement Learning that Matters, 2017

What is measured?

Average or median reward?

Cumulative performance while learning (regret)?

Performance with exploration, or separate evaluation where exploration is “turned of”?

Different choices are justifiable, but it should at least be clear *what* was chosen, and preferably *why*!

What is reported?

Final performance or whole learning curve?

- Whole learning curve shows learning speed as well as performance. Preferable unless there are many results to report…

What do error bars indicate

- 1 or 2 std dev, std error, interquartile range
- independently trained models or multiple runs of same model?

  Most deep learning models really need to be trained multiple times to get a reliable result!

At least make clear *what* was chosen, and preferably *why*!

Recommendations

Run multiple trials with different random seeds, starting from scratch each time.

Report mean/median as well as a measure of ‘spread’

Sanity check result of your hyperparameters & implementation with published results

Spend roughly equal amount of effort tuning different methods

Report all details required for reproduction and interpretation!

- All hyperparameters, architecture, ‘tricks’
- What was measured, how, #of independent runs

Debugging RL methods

SOTA RL methods have many different parts, and it can be very tricky to find out why something doesn't work.

A lot of 'making things work' comes with experience. Some general tips:

- Try to localise errors. Do separate modules (actor, critic) yield the desired result? Can you do a part of the problem (e.g. policy evaluation) stand alone first?
- Running (part of) the algorithm on a fixed dataset (in 'batch mode') avoids feedback loops
- Can you plug-in your module with modules that are known to work decently? (swap out modules with working code)

Debugging RL methods

Since the current q-network / policy network / etc influences what data is gathered, the loss doesn't always decrease

- contrast with e.g. MNIST classification, where the data is fixed and you expect the model to get better over time
- To debug, can use 'batch mode' to see whether learning is effective
- another possibility is freezing either the actor or the critic while training the other

"Actor loss" is not a regular loss function

- Since $\boxed{\nabla_{\boldsymbol{\theta}}(G_t \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}))} = \boxed{G_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})}$
  $G_t \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$ used as 'actor loss' with autodiff frameworks
- This is a proxy with the same gradient, but different higher-order terms and value then the real objective $\mathbb{E}[J(\theta)]$

What you should know

What is the main concept behind DPG?

What are some challenges when tackling reinforcement learning problems using deep neural networks?

What are some main challenges in evaluating deep reinforcement learning methods?

How to recognise and avoid problems in evaluating deep reinforcement learning methods?

Any questions?

Feedback?

h.c.vanhoof@uva.nl