DDPG versus CMA-ES: a detailed comparison

Olivier Sigaud
http://people.isir.upmc.fr/sigaud

Joint work with Arnaud de Froissard de Broissia

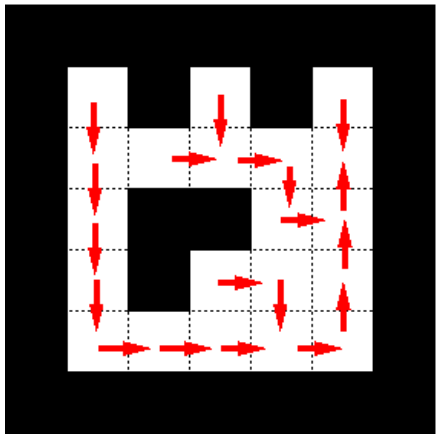
April 20, 2016

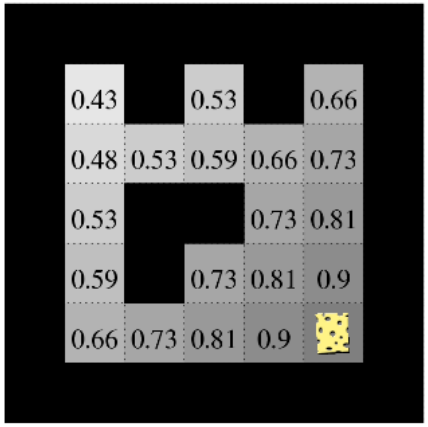
ISIR
INSTITUT
DES SYSTÈMES
INTELLIGENTS
ET DE ROBOTIQUE

Outline

- Some background about actor-critic
- Motivate the comparison to CMA-ES
- What is DDPG, how does it work?
- Detailed comparison of the mechanisms
- Experimental study
- Discussion, conclusions

Policy and value functions





state / action	a_0	a_1	a_2	a_3
e_0	0.66	0.88	0.81	0.73
e_1	0.73	0.63	0.9	0.43
e_2	0.73	0.9	0.95	0.73
e_3	0.81	0.9	1.0	0.81
e_4	0.81	1.0	0.81	0.9
e_5	0.9	1.0	0.0	0.9

- Goal: find a policy $\pi : S \to A$ maximizing the agregation of reward on the long run
- The value function $V^\pi : S \to \mathbb{R}$ records the agregation of reward on the long run for each state (following policy π). It is a vector with one entry per state
- The action value function $Q^\pi : S \times A \to \mathbb{R}$ records the agregation of reward on the long run for doing each action in each state (and then following policy π). It is a matrix with one entry per state and per action

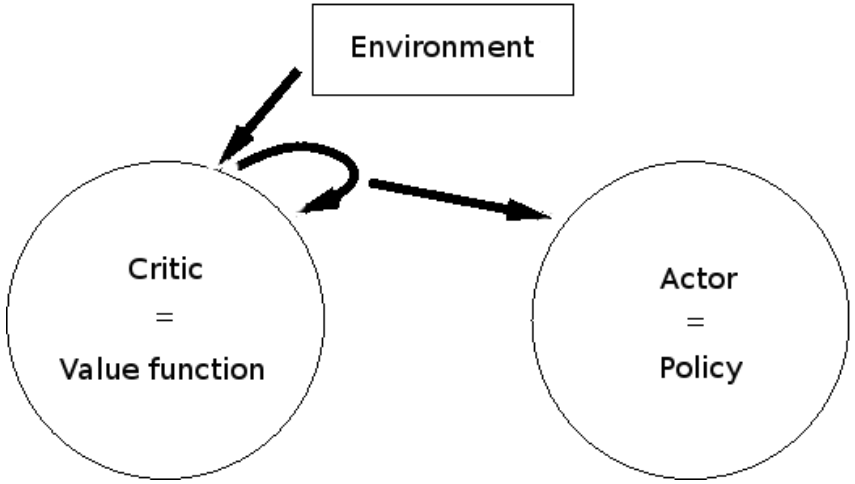
Families of methods

- Critic : (action) value function $\rightarrow$ evaluation of the policy
- Actor: the policy itself
- Critic-only methods: iterates on the value function up to convergence without storing policy, then computes optimal policy. Typical examples: value iteration, Q-learning, Sarsa
- Actor-only methods: explore the space of policy parameters. Typical example: CMA-ES
- Actor-critic methods: update in parallel one structure for the actor and one for the critic. Typical examples: policy iteration, many AC algorithms
- Q-learning and Sarsa look for a global optimum, AC looks for a local one

Temporal Difference error

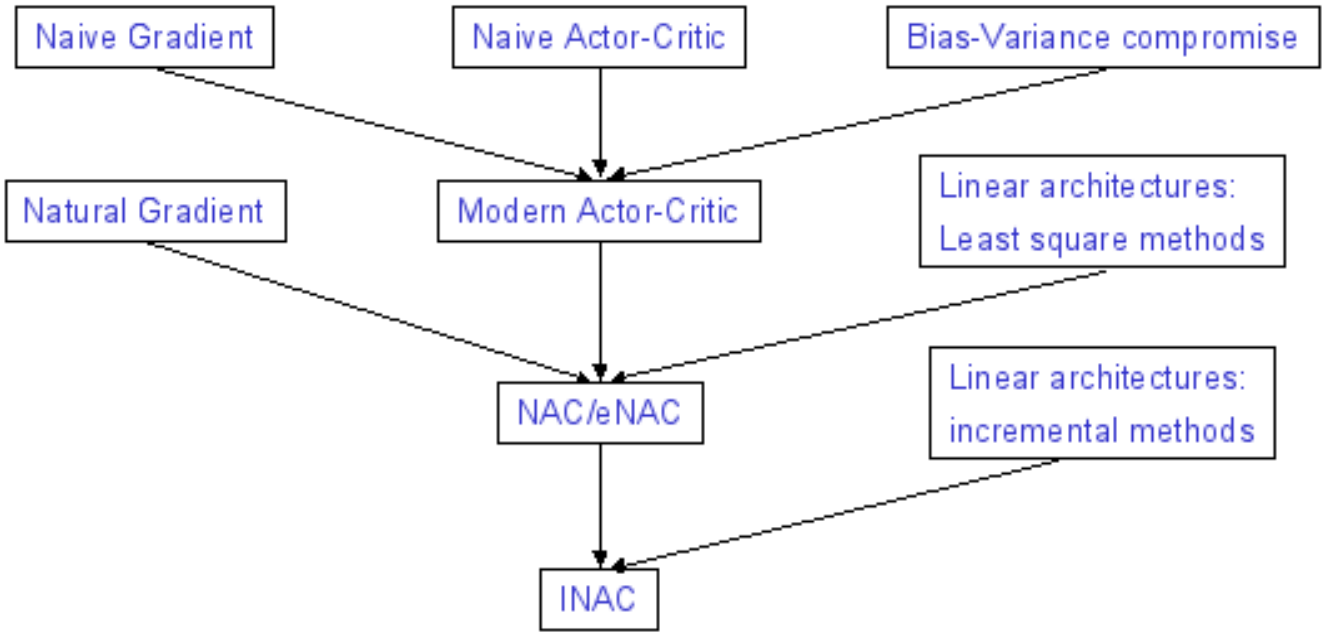
- The goal of TD methods is to estimate the value function $V(s)$
- If estimations $V(s_t)$ and $V(s_{t+1})$ were exact, we would get:
- $V(s_t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ...$
- $V(s_{t+1}) = r_{t+2} + \gamma(r_{t+3} + \gamma^2 r_{t+4} + ...$
- Thus $V(s_t) = r_{t+1} + \gamma V(s_{t+1})$
- $\delta_k = r_{k+1} + \gamma V(s_{k+1}) - V(s_k)$: measures the error between current values of V and the values they should have
- If δ positive, increase V, if negative, decrease V

Naive actor-critic approach



- Discrete states and actions, stochastic policy
- An update in the critic generates a local update in the actor
- Critic: compute δ and update $V(s)$ with $V_k(s) \leftarrow V_k(s) + \alpha_k \delta_k$
- Actor: $P^\pi(a|s) = P^\pi(a|s) + \alpha_k \prime \delta_k$
- NB: no need for a max over actions, but local maximum
- NB2: one must know how to "draw" an action from a probabilistic policy (not straightforward for continuous actions)

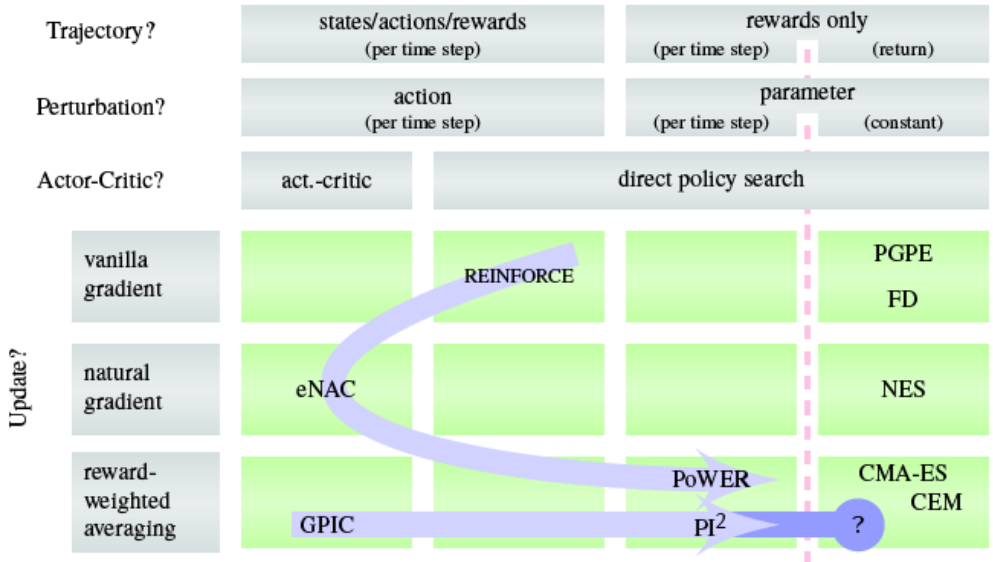
Quick history



- Those methods proved inefficient for robot RL

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000) Policy gradient methods for reinforcement learning with function approximation. In NIPS 12 (pp. 1057–1063).: MIT Press.

Motivation: why compare to CMA-ES?



- Towards "blind" policy search (CMA-ES) + DMPs (small domain)
- Requires DMP engineering
- In principle, actor-critic should be more data efficient
- But sensitive to value function approximation error
- DDPG brings accurate value function approximation and no feature engineering

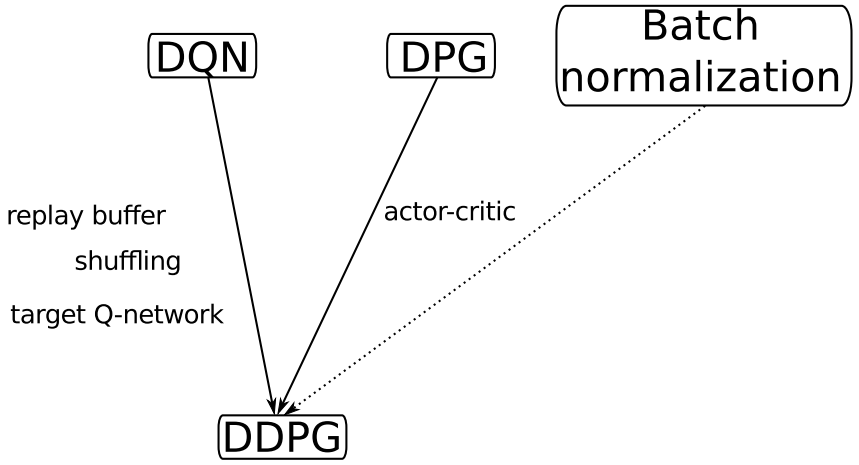
Stulp, F. & Sigaud, O. (2012) Path integral policy improvement with covariance matrix adaptation, In *Proceedings ICML 29* (pp. 1–8). Edinburgh, Scotland.

DDPG: The paper

- Continuous control with deep reinforcement learning
- Timothy P. Lillicrap Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra
- Google Deepmind
- On arXiv since september 7, 2015
- Already cited 21 times

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971 7/9/15*.

DDPG: ancestors

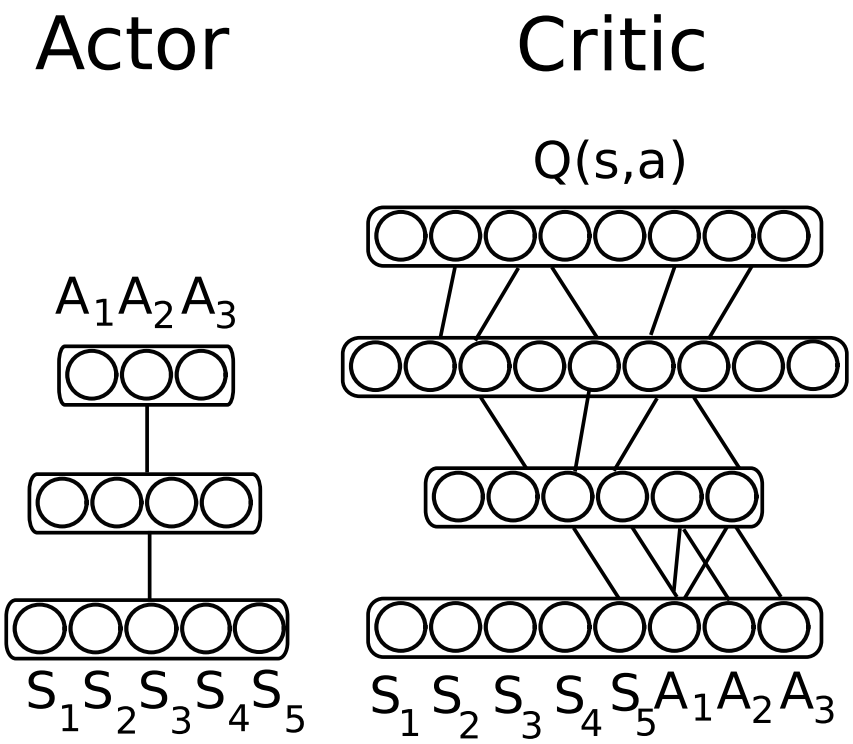


- DQN: Atari domain, Nature paper, small discrete actions set
- Most of the actor-critic theory for continuous problem is for stochastic policies (policy gradient theorem, compatible features, etc.)

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015) Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

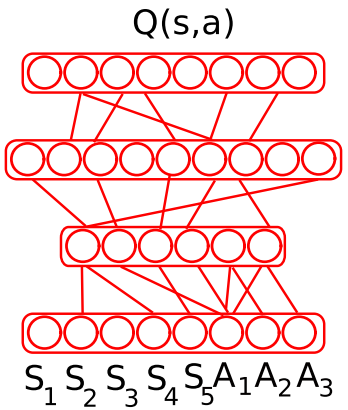
Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014) Deterministic policy gradient algorithms. In *ICML*.

General architecture



- Any neural network structure
- Actor parametrized by $\mathbf{w}$, critic by θ

Training the critic



- In DPG (and RL in general), the critic should minimize the RPE:

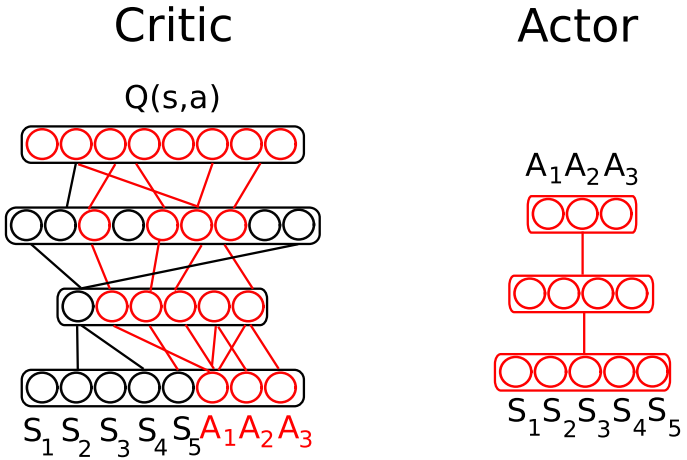
$$\delta = r + \gamma Q(s_{t+1}, \pi(s_{t+1})|\theta) - Q(s_t, a_t|\theta)$$

- We want to minimize the critic error using backprop on critic weights θ
- Error = difference between network output $Q(s_t, a_t|\theta)$ and "something"
- Thus, given N samples $\{s_i, a_i, r_i, s_{i+1}\}$, compute $y_i = r_i + \gamma Q(s_{t+1}, \pi(s_{t+1})|\theta)$
- And update θ by minimizing the squared loss function (backpropagation of the squared error) over the batch

$$L = 1/N \sum_i (y_i - Q(s_i, a_i|\theta))^2 \quad (1)$$

- Backprop is available in TensorFlow, theano... (RProp, RMSProp, Adagrad, Adam?)

Training the actor



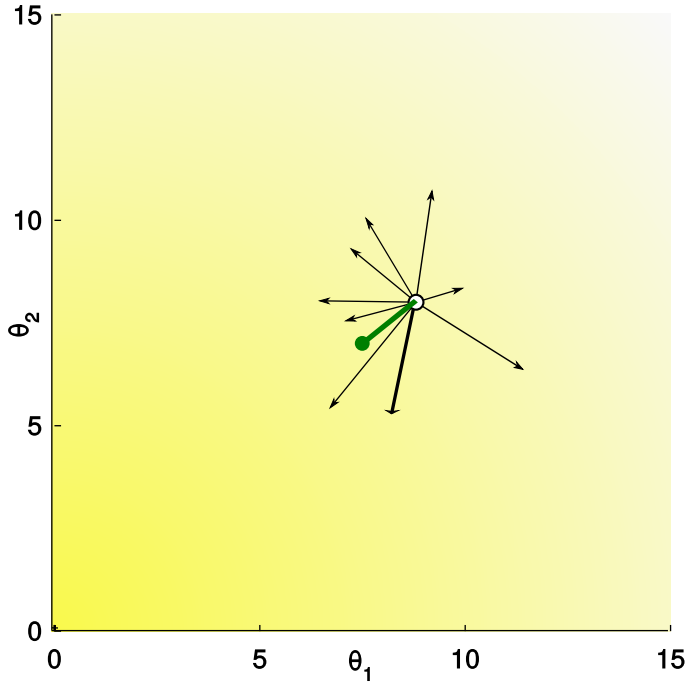
- Deterministic policy gradient theorem: the true policy gradient is

$$\nabla_{\mathbf{w}}\pi(s,a) = \mathbb{E}_{\rho(s)}[\nabla_a Q(s,a|\theta)\nabla_{\mathbf{w}}\pi(s|\mathbf{w})] \quad (2)$$

- $\nabla_a Q(s,a|\theta)$ is obtained by computing the gradient over actions of $Q(s,a|\theta)$ in the critic.
- The gradient over actions is similar to the gradient over weights (symmetric roles of weights and inputs)
- $\nabla_a Q(s,a|\theta)$ is used as an error signal to update the actor's weights through backprop again.
- Comes from NFQCA

Hafner, R. & Riedmiller, M. (2011) Reinforcement learning in feedback control. *Machine learning*, 84(1-2), 137–169.

Actor update over a batch



- Consider a batch of N samples
- Compute the gradient of the critic for each sample
- Then compute the corresponding gradient of the actor
- Compute the average over all these gradients (all weights to 1)
- Defines the new actor parameters

General algorithm

1. Feed the actor with the state, outputs the action
2. Feed the critic with the state and action, determines $Q(s, a|\theta^Q)$
3. Update the critic, using (1) (alternative: do it after 4?)
4. Compute $\nabla_a Q(s, a|\theta)$
5. Update the actor, using (2)

Subtleties

- The actor update rule is

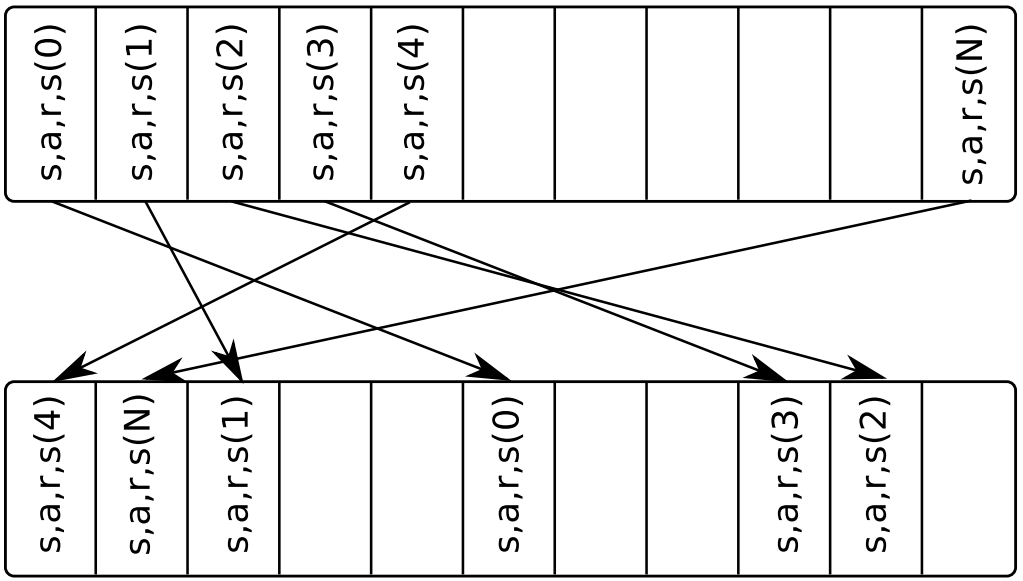
$$\nabla_{\mathbf{w}}\pi(s_i) \approx 1/N \sum_i \nabla_a Q(s,a|\theta)|_{s=s_i,a=\pi(s_i)} \nabla_{\mathbf{w}}\pi(s)|_{s=s_i}$$

- Thus we do not use the action in the samples to update the actor
- Could it be

$$\nabla_{\mathbf{w}}\pi(s_i) \approx 1/N \sum_i \nabla_a Q(s,a|\theta)|_{s=s_i,a=a_i} \nabla_{\mathbf{w}}\pi(s)|_{s=s_i}?$$

- Work on $\pi(s_i)$ instead of a_i
- Does this make the algorithm on-policy instead of off-policy?
- Does this make a difference?

Trick 1: Sample buffer (from DQN)



- In most optimization algorithms, samples are assumed independently and identically distributed (iid)
- Obviously, this is not the case of behavioral samples (s_i, a_i, r_i, s_{i+1})
- Idea: put the samples into a buffer, and extract them randomly

Trick 2: Stable Target Q-function (from DQN)

- Compute the critic loss function from a separate target network $Q'(...|\theta')$
- So compute $y_i = r_i + \gamma Q'(s_{i+1}, \pi(s_{i+1})|\theta')$
- In DQN, the θ is updated after each batch
- In DDPG, they rather allow for slow evolution of Q' and π'

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

- The same applies to μ, μ'
- From the empirical study, this is the critical trick

Trick 3: Batch Normalization (new)

- Covariate shift: as layer N is trained, the input distribution of layer $N+1$ is shifted, which makes learning harder
- To fight covariate shift, ensure that each dimension across the samples in a minibatch have unit mean and variance at each layer
- Add a buffer between each layer, and normalize all samples in these buffers
- Makes learning easier and faster
- Makes the algorithm more domain-insensitive
- But poor theoretical grounding, and makes network computation slower

Ioffe, S. & Szegedy, C. (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Algorithm

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 gradient:

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

    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

- Notice the slow Q' and π' updates (instead of copying as in DQN)

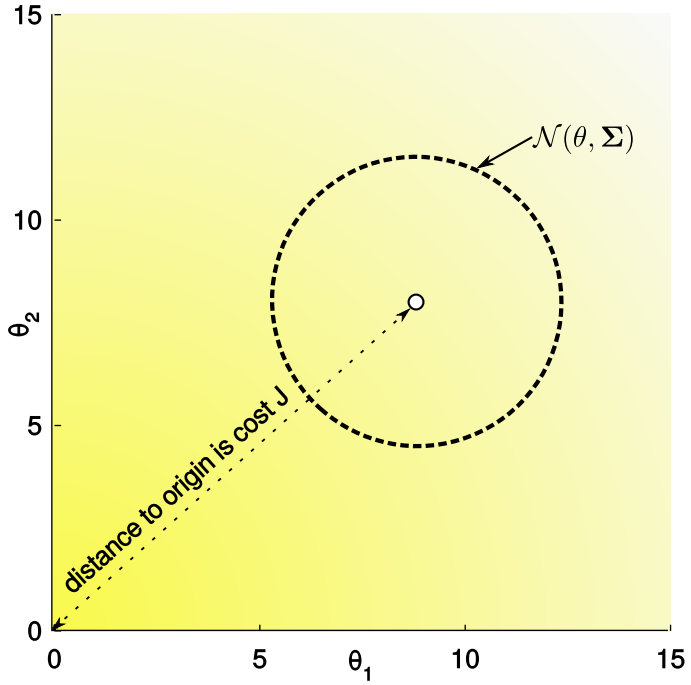
Applications: impressive results

- End-to-end policies (from pixels to control)
- Works impressively well on "More than 20" (27-32) such domains
- Coded with MuJoCo (Todorov) / TORCS

Motivation of the comparison

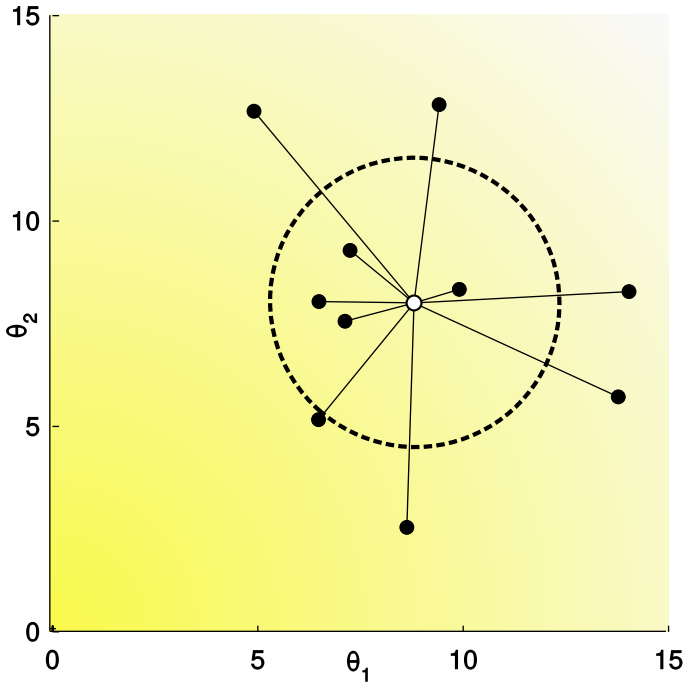
- DDPG still looks for a local minimum, like any actor-critic method
- In contrast to CMA-ES, DDPG can learn a policy with many parameters (CMA-ES on actor with 2 hidden layers of 1000 neurons: "out of memory"!)
- DDPG is mostly useful to learn "end-to-end" policies
- But does value function approximation provide a clear advantage wrt to actor-only policy search?
- Unfair comparison: DDPG is outside its application domain (small actor network)
- Is it still better than CMA-ES?
- If yes, stop using CMA-ES...

Set-up: initialization



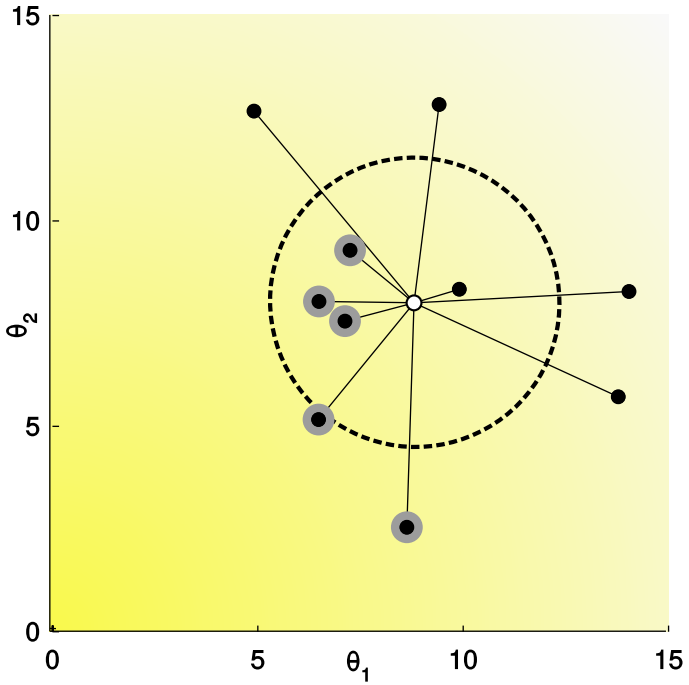
- Define an initial covariance matrix

Generate samples



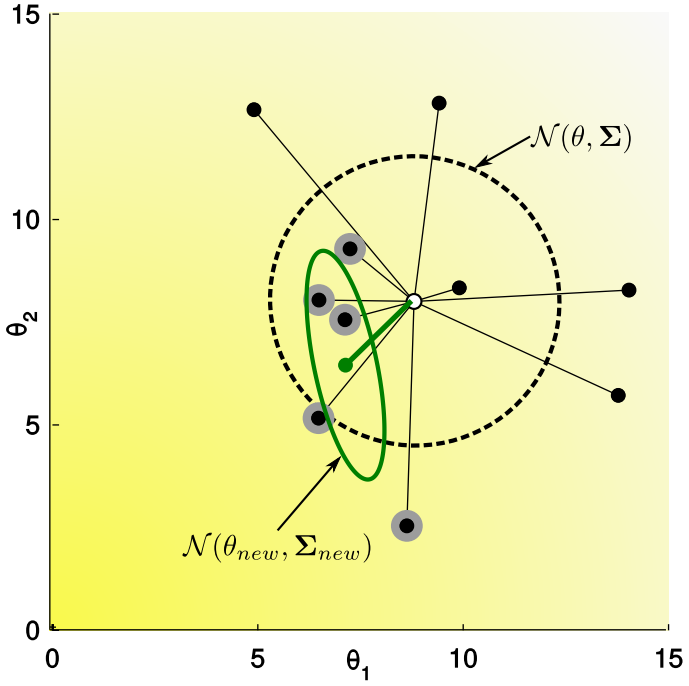
- A fixed number N of samples are drawn according to the current covariance matrix

Evaluate samples



- This is the costly step: evaluate the actor performing complete trajectories
- If the problem is stochastic, several trajectories are required per sample
- Evaluation at the end of each trajectory

Reward weighted averaging



- Compute new parameter vector as an average of the samples weighted by their performance
- Then adapt the covariance matrix

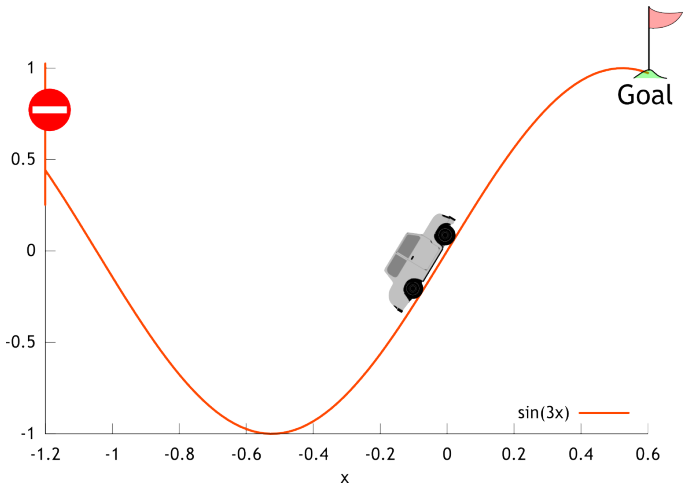
CMA-ES vs DDPG: comparison

- Computation of the gradient in DDPG does just require the N samples
- Compared to $k \times N$ trajectories
- Samples at each step versus evaluation at the end of the trajectory
- The return in CMA-ES is exact, whereas in DDPG it depends on the accuracy of the critic
- Samples for critic training versus samples for policy improvement

Closer comparison

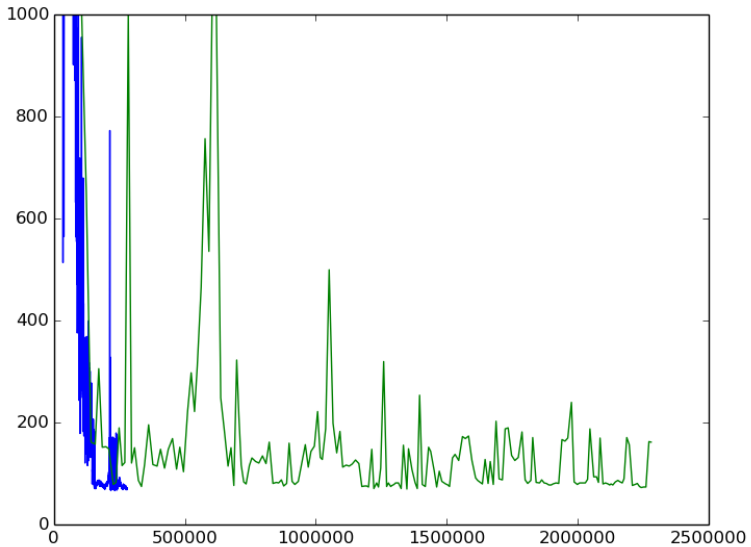
- Source of randomness in CMA-ES: drawing the samples
- Source of randomness in DDPG: exploration noise in the policy, stochastic gradient descent (?)
- Exploration noise in the policy: in DDPG, action perturbation rather than policy parameter perturbation
- Rather try the latter?
- The gradient of DDPG makes analytic profit of knowing the structure of the actor (function derivation)
- Can we express the gradient of the actor in DDPG in terms of equivalent trajectory samples?
- Using the derivative as a first order difference $\frac{(\theta - \theta')}{d_t}$?
- What about performing weighted average instead of all samples weighted to 1?

Work in progress...



- Cost = squared acceleration per time step
- Reward if goal reached
- Based on the mountain car benchmark
- Very simple actor: two input, one output, 2 hidden layers with 20? and 10? neurons respectively
- No batch normalization nor weight normalization

Very preliminary results



- X = number of calls to simulator, Y = number of steps to reach goal
- The time to compute both results is similar
- This (seems to) illustrate that DDPG is much more sample efficient
- To be confirmed by comparing time, averaging over more results, etc.
- Also compare using batch norm, weight norm, etc.

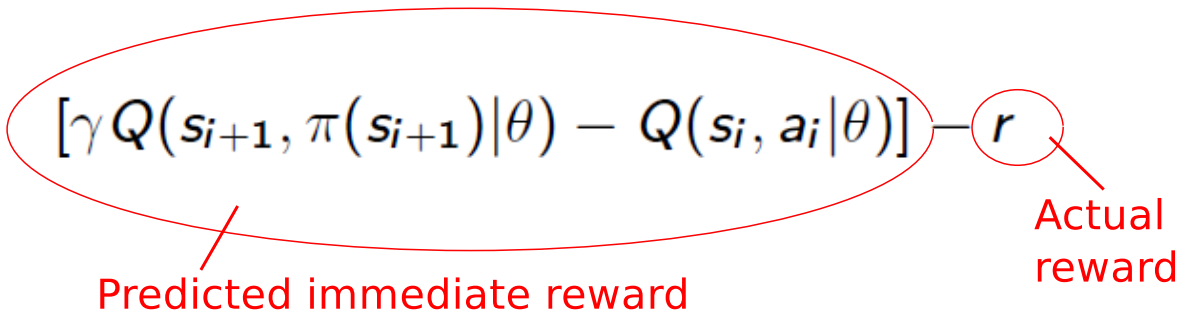
Comparison: next steps

- Try a steeper mountain
- Given the same actor and the same set of samples, compare the progress you get in the actor between CMA-ES and DDPG:
  - with a naive critic
  - with a midly trained critic
  - with an optimal critic
- Compare to Bayesian optimization
- Apply it to more challenging robot learning benchmark

CMA-ES first, then DDPG

- When the critic is not good, CMA-ES may work better than DDPG
- But once the critic is good, DDPG is more efficient
- So CMA-ES may work better than DDPG in the beginning, and much slower then
- Try to switch from a CMA-ES-like approach to a DDPG-like approach along time

Approximate the immediate reward



- The new information in the RPE update rule is the reward,
- The critic network may encode the expected immediate reward function $f_\theta(s_i, a_i, s_{i+1}) = \gamma Q(s_{i+1}, \pi(s_{i+1})|\theta) - Q(s_i, a_i|\theta)$
- We still get δ, but we don't know how to compute $\nabla_a Q(s, a|\theta)$

Approximate the advantage function

- Other option: encode the advantage function
  $A_\theta(s_i, a_i) = Q(s_i, a_i|\theta) - max_a Q(s_i, a|\theta)$
- Very good recent paper
- Or see GProp...

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016) Continuous deep q-learning with model-based acceleration, *arXiv preprint arXiv:1603.00748*.

Balduzzi, D. and Ghifary, M. (2015). Compatible value gradients for reinforcement learning of continuous deep policies, *arXiv preprint arXiv:1509.03005 10/9/15*.

Better exploration

- DDPG does not help to find scarce rewards (the needle in the stack): no specific exploration
- Get inspired by diversity search in evolutionary techniques

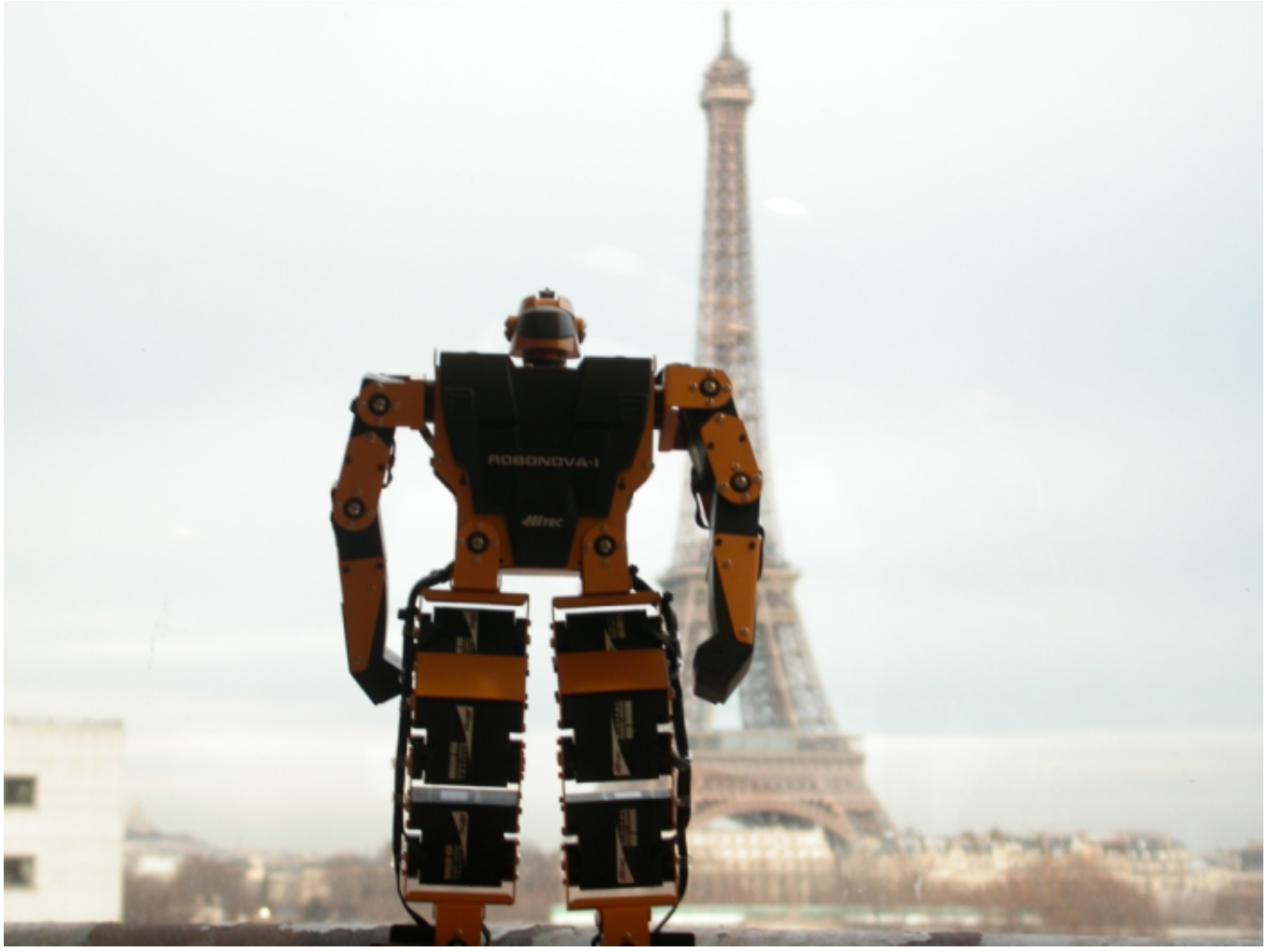
Back to natural gradient

- Batch normalization
- Weight normalization
- Natural Neural networks

Salimans, T. & Kingma, D. P. (2016) Weight normalization: A simple reparameterization to accelerate training of deep neural networks, *arXiv preprint arXiv:1602.07868*.

Desjardins, G., Simonyan, K., Pascanu, R., et al. (2015) Natural neural networks, In *Advances in Neural Information Processing Systems* (pp. 2062–2070).

Any question?



Balduzzi, D. (2015).

Semantics, representations and grammars for deep learning.
arXiv preprint arXiv:1509.08627.

Balduzzi, D. & Ghifary, M. (2015).

Compatible value gradients for reinforcement learning of continuous deep policies.
arXiv preprint arXiv:1509.03005.

de Bruin, T., Kober, J., Tuyls, K., & Babuška, R.

The importance of experience replay database composition in deep reinforcement learning.
In *Deep RL workshop at NIPS 2015*.

Desjardins, G., Simonyan, K., Pascanu, R., et al. (2015).

Natural neural networks.
In *Advances in Neural Information Processing Systems* (pp. 2062–2070).

Dulac-Arnold, G., Evans, R., Sunehag, P., & Coppin, B. (2015).

Reinforcement learning in large discrete action spaces.
arXiv preprint arXiv:1512.07679.

Fragkiadaki, K., Agrawal, P., Levine, S., & Malik, J. (2015).

Learning visual predictive models of physics for playing billiards.
arXiv preprint arXiv:1511.07404.

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016).

Continuous deep q-learning with model-based acceleration.
arXiv preprint arXiv:1603.00748.

Hafner, R. & Riedmiller, M. (2011).

Reinforcement learning in feedback control.
Machine learning, 84(1-2), 137–169.

Hausknecht, M. & Stone, P. (2015).

Deep reinforcement learning in parameterized action space.
arXiv preprint arXiv:1511.04143.

Heess, N., Hunt, J. J., Lillicrap, T. P., & Silver, D. (2015a).

Memory-based control with recurrent neural networks.
arXiv preprint arXiv:1512.04455.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., & Tassa, Y. (2015b).

Learning continuous control policies by stochastic value gradients.
In *Advances in Neural Information Processing Systems* (pp. 2926–2934).

Ioffe, S. & Szegedy, C. (2015).

Batch normalization: Accelerating deep network training by reducing internal covariate shift.
arXiv preprint arXiv:1502.03167.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015).

Continuous control with deep reinforcement learning.
arXiv preprint arXiv:1509.02971.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015).
Human-level control through deep reinforcement learning.
Nature, 518(7540), 529–533.

Parisotto, E., Ba, J. L., & Salakhutdinov, R. (2015).

Actor-mimic: Deep multitask and transfer reinforcement learning.
arXiv preprint arXiv:1511.06342.

Salimans, T. & Kingma, D. P. (2016).

Weight normalization: A simple reparameterization to accelerate training of deep neural networks.
arXiv preprint arXiv:1602.07868.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014).

Deterministic policy gradient algorithms.
In *Proceedings of the 30th International Conference in Machine Learning*.

Stulp, F. & Sigaud, O. (2012).

Path integral policy improvement with covariance matrix adaptation.
In *Proceedings of the 29th International Conference on Machine Learning (ICML)* (pp. 1–8). Edinburgh, Scotland.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000).

Policy gradient methods for reinforcement learning with function approximation.
In *Advances in Neural Information Processing Systems 12* (pp. 1057–1063).: MIT Press.

Tzeng, E., Devin, C., Hoffman, J., Finn, C., Peng, X., Levine, S., Saenko, K., & Darrell, T. (2015).

Towards adapting deep visuomotor representations from simulated to real environments.
arXiv preprint arXiv:1511.07111.

Yoshida, N. (2015).

Q-networks for binary vector actions.
arXiv preprint arXiv:1512.01332.

Computational neuroscience impact

- Deep learning models of cortical computations (ConvNets, etc.)
- Deep RL: cortex + basal ganglia?
- Bird song as the ideal domain

Testing DDPG

- Tests DDPG, in particular experience replay database
- Does not work everytime...

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. (2015) The importance of experience replay database composition in deep reinforcement learning.

See: http://rll.berkeley.edu/deeprlworkshop/ (23 papers)

Compatible value gradient (GProp)

- Clean (difficult?) theoretical paper referenced in 4 others
- Approx the value function gradient rather than derive the approx of the value function (by adding Gaussian noise to the function)
- Value gradient via backpropagation
- Better than supervised learning on SARCOS/BARRETT via a bandit formulation (very clear)

Balduzzi, D. and Ghifary, M. (2015). Compatible value gradients for reinforcement learning of continuous deep policies. *arXiv preprint arXiv:1509.03005 10/9/15*.

Stochastic Value Gradient

- Extends DDPG to stochastic value gradients
- Difficult NIPS paper, poorly explained
- Cites [Balduzzi & Ghifary, 2015], but follows a different line

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. (2015) Learning continuous control policies by stochastic value gradients, In *Advances in Neural Information Processing Systems*, pages 2926–2934. (˜ 15/12/15).

Actor-Mimic

- Transfer learning on Atari games
- Learns individual experts on each game
- Builds a "multi-game" expert
- Learns faster on a new game

Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342 6/1/16*.

Hausknecht et Stone

- Application of DDPG to RoboCup
- Uses structured parametrized actions (list with continuous parameters)
- Usefulness : (sometimes dubious?) programming-centered presentation

Hausknecht, M. and Stone, P. (2015). Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143 8/1/16*.

Memory-based control

- Extends DDPG to POMDPs (use of memory)
- Combines DDGP and LSTM
- Applications: cart-pole without speed, watermaze
- Very few details on applications
- Follow-up: use GRU, Clockwork-RNN, SRCN?

Heess, N., Hunt, J. J., Lillicrap, T. P., and Silver, D. (2015). Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455 14/12/15*.

Results: learning curves

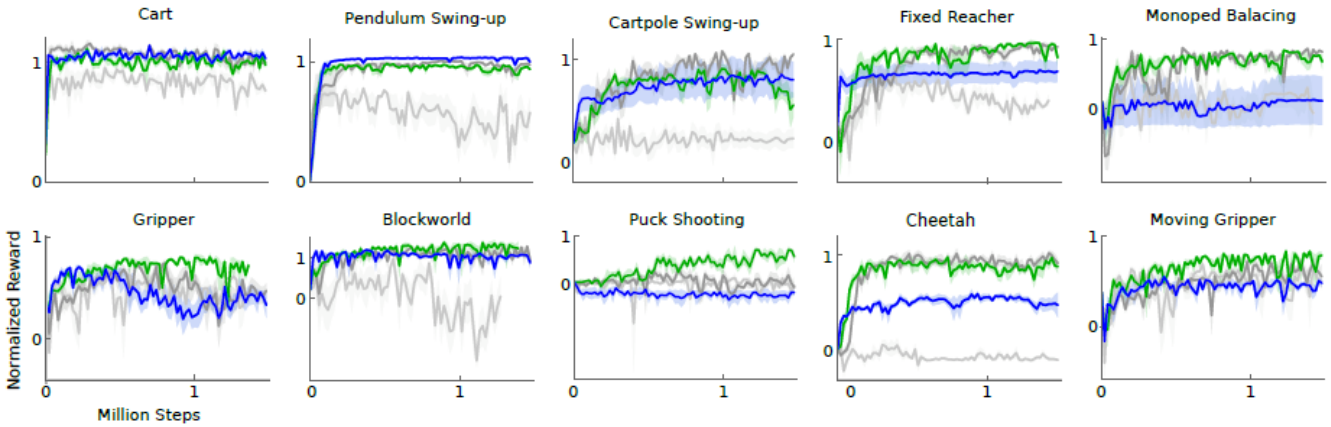


Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target networks are crucial.

- Notice the million steps!

Results: performance

environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$
blockworld1	1.156	1.511	0.466	1.299
blockworld3da	0.340	0.705	0.889	2.225
canada	0.303	1.735	0.176	0.688
canada2d	0.400	0.978	-0.285	0.119
cart	0.938	1.336	1.096	1.258
cartpole	0.844	1.115	0.482	1.138
cartpoleBalance	0.951	1.000	0.335	0.996
cartpoleParallelDouble	0.549	0.900	0.188	0.323
cartpoleSerialDouble	0.272	0.719	0.195	0.642
cartpoleSerialTriple	0.736	0.946	0.412	0.427
cheetah	0.903	1.206	0.457	0.792
fixedReacher	0.849	1.021	0.693	0.981
fixedReacherDouble	0.924	0.996	0.872	0.943
fixedReacherSingle	0.954	1.000	0.827	0.995
gripper	0.655	0.972	0.406	0.790
gripperRandom	0.618	0.937	0.082	0.791
hardCheetah	1.311	1.990	1.204	1.431
hopper	0.676	0.936	0.112	0.924
hyq	0.416	0.722	0.234	0.672
movingGripper	0.474	0.936	0.480	0.644
pendulum	0.946	1.021	0.663	1.055
reacher	0.720	0.987	0.194	0.878
reacher3daFixedTarget	0.585	0.943	0.453	0.922
reacher3daRandomTarget	0.467	0.739	0.374	0.735
reacherSingle	0.981	1.102	1.000	1.083
walker2d	0.705	1.573	0.944	1.476
torcs	-393.385	1840.036	-401.911	1876.284

- Sometimes better (>1) than iLQG (model-based, pure planning)
- Results on TORCS are not so good

Results: analysis

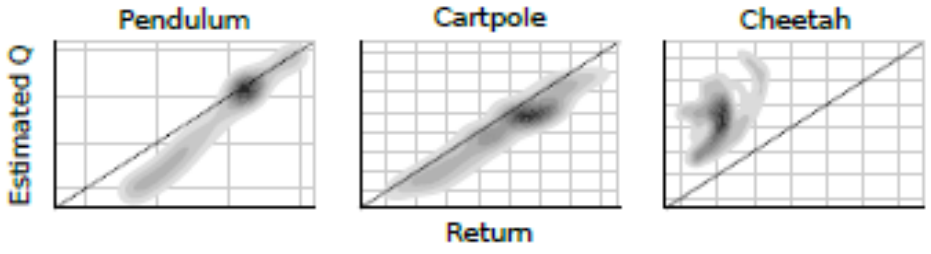


Figure 3: Density plot showing estimated Q values versus observed returns sampled from test episodes on 5 replicas. In simple domains such as pendulum and cartpole the Q values are quite accurate. In more complex tasks, the Q estimates are less accurate, but can still be used to learn competent policies. Dotted line indicates unity, units are arbitrary.

- The closer to the diagonal, the better the estimate
- It is not always so good
- So why does it work so well?

Following papers (1)

- Balduzzi, D. and Ghifary, M. (2015). Compatible value gradients for reinforcement learning of continuous deep policies. *arXiv preprint arXiv:1509.03005 10/9/15*.
- Hausknecht, M. and Stone, P. (2015). Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143 8/1/16*.
- Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342 6/1/16*.
- Heess, N., Hunt, J. J., Lillicrap, T. P., and Silver, D. (2015). Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455 14/12/15*.
- Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. (2015). Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2926–2934. (˜ 15/12/15)
- de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. The importance of experience replay database composition in deep reinforcement learning. Deep RL workshop, NIPS 15

Following papers (2)


Balduzzi, D. (2015). Semantics, representations and grammars for deep learning. *arXiv preprint arXiv:1509.08627 29/9/15*.


Dulac-Arnold, G., Evans, R., Sunehag, P., and Coppin, B. (2015). Reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679 24/12/15*.


Yoshida, N. (2015). Q-networks for binary vector actions. *arXiv preprint arXiv:1512.01332*.


Tzeng, E., Devin, C., Hoffman, J., Finn, C., Peng, X., Levine, S., Saenko, K., and Darrell, T. (2015). Towards adapting deep visuomotor representations from simulated to real environments. *arXiv preprint arXiv:1511.07111*.


Fragkiadaki, K., Agrawal, P., Levine, S., and Malik, J. (2015). Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*.

See also : http://rll.berkeley.edu/deeprlworkshop/ (23 papers)

ISIR
INSTITUT
DES SYSTÈMES
INTELLIGENTS
ET DE ROBOTIQUE