Reinforcement Learning

From the basics to Deep RL

Olivier Sigaud

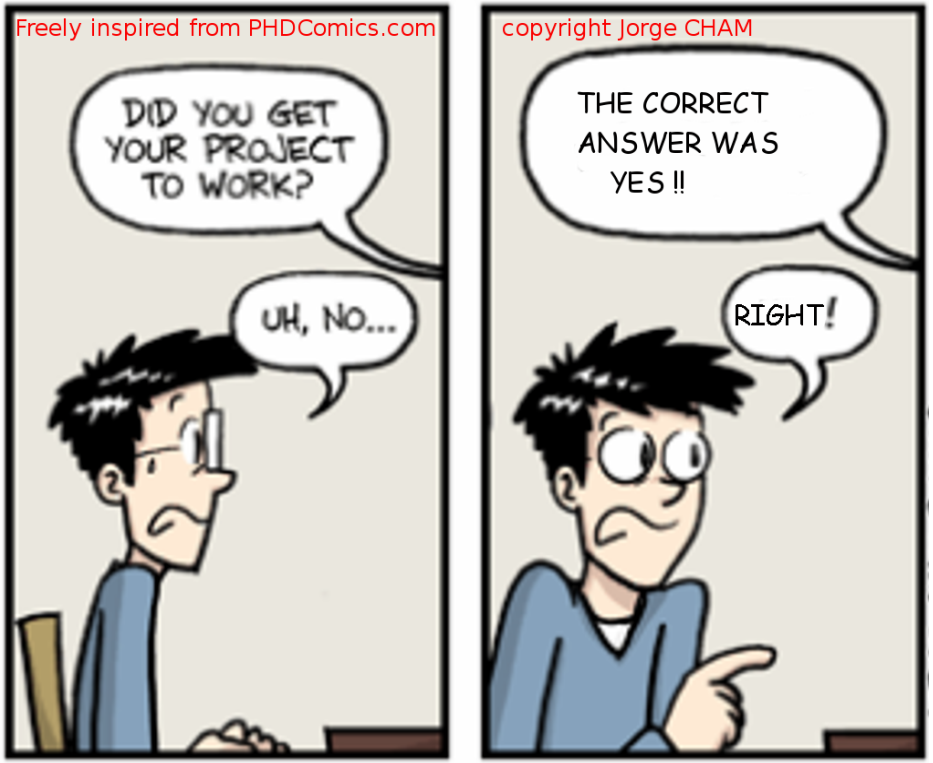
ISIR, UPMC + INRIA
http://people.isir.upmc.fr/sigaud

September 14, 2017

Outline

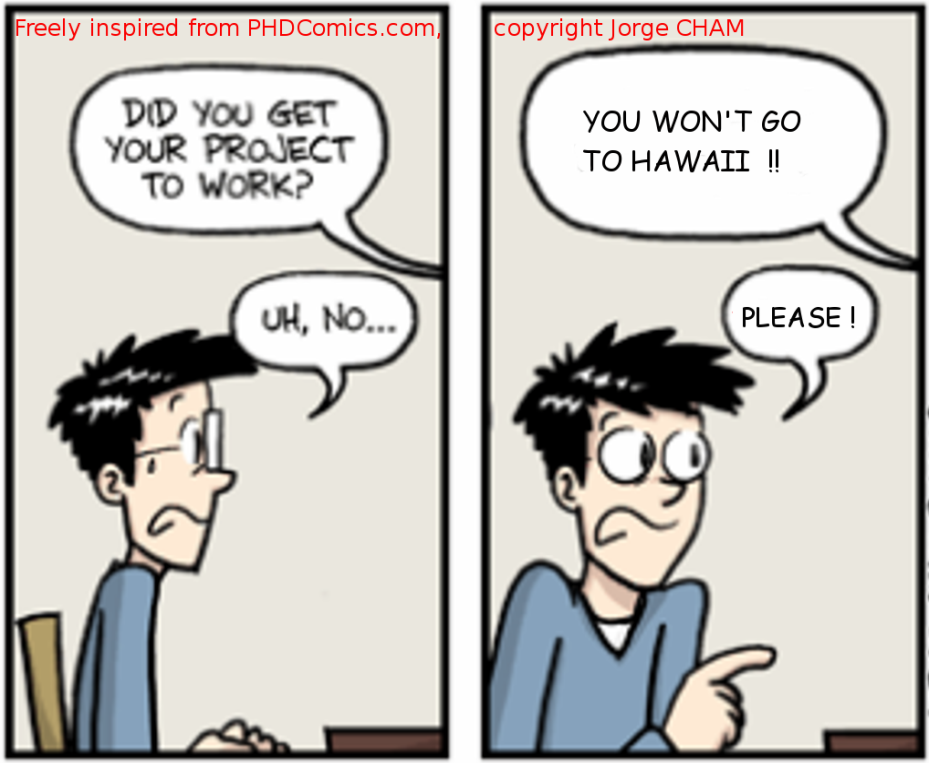
- Some quick background about discrete RL and actor-critic methods
- DQN and the main tricks
- Beyond DQN: a few state-of-the-art papers
- What is DDPG, how does it work?
- Further algorithms: NAF, TRPO, ...

Supervised learning



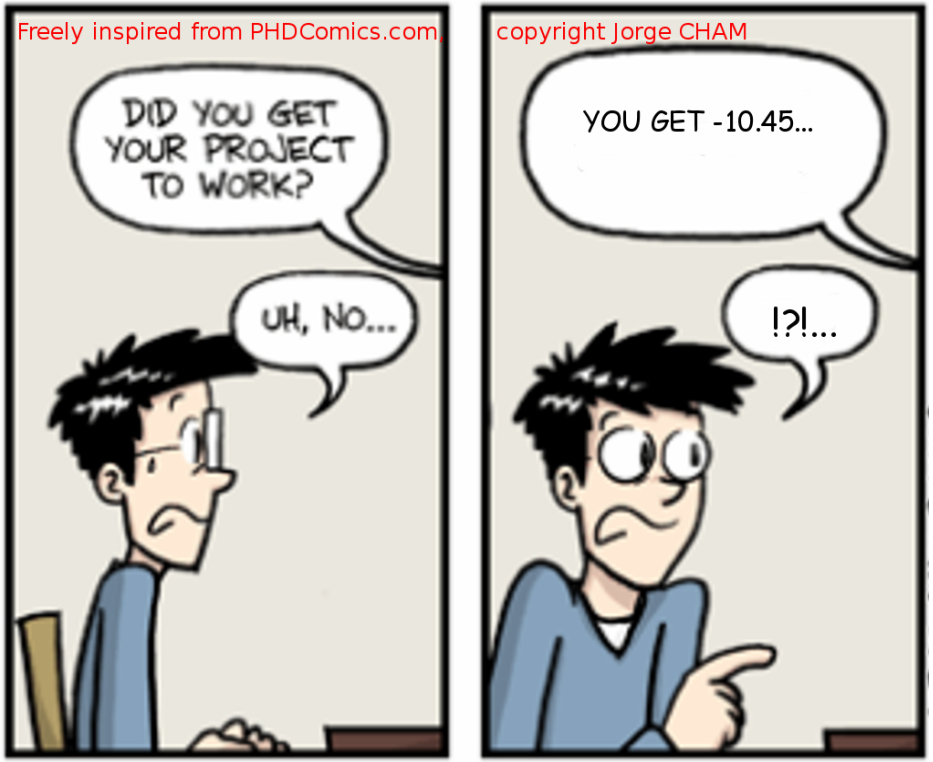
- The supervisor indicates to the agent the expected answer
- The agent corrects a model based on the answer
- Typical mechanism: gradient backpropagation, RLS
- Applications: classification, regression, function approximation...

Cost-Sensitive Learning



- The environment provides the value of action (reward, penalty)
- Application: behaviour optimization

Reinforcement learning



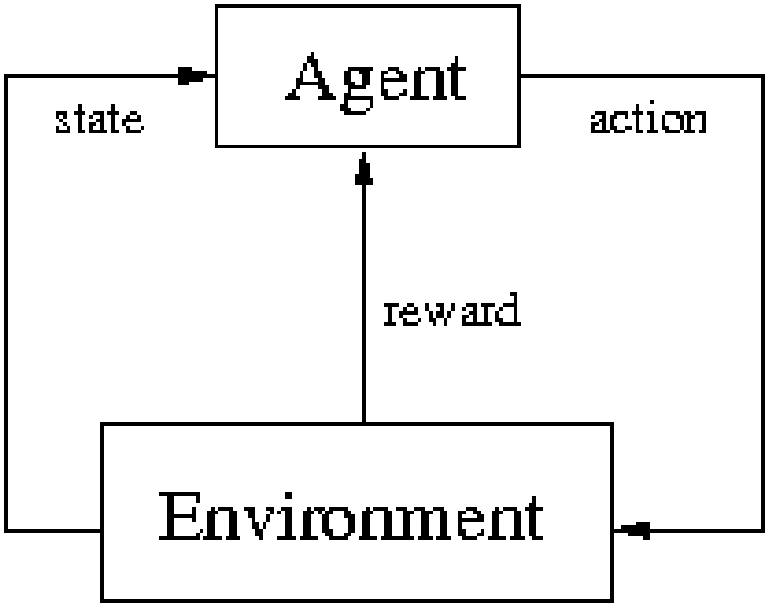
- In RL, the value signal is given as a scalar
- How good is -10.45?
- Necessity of exploration

The exploration/exploitation trade-off

- Exploring can be (very) harmful
- Shall I exploit what I know or look for a better policy?
- Am I optimal? Shall I keep exploring or stop?
- Decrease the rate of exploration along time
- ϵ-*greedy*: take the best action most of the time, and a random action from time to time

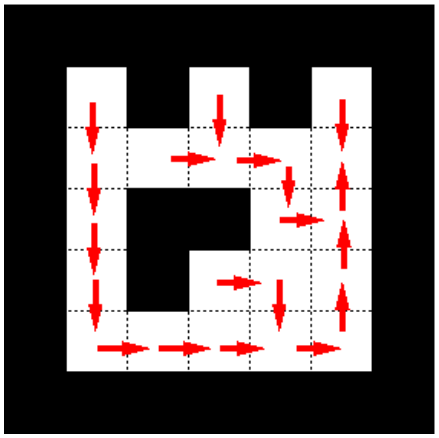
Markov Decision Processes

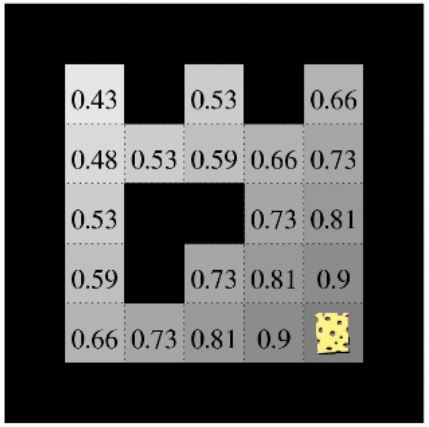
- S: states space
- A: action space
- $T : S \times A \rightarrow \Pi(S)$: transition function
- $r : S \times A \rightarrow \mathbb{R}$: reward function



- An MDP defines s^{t+1} and r^{t+1} as $f(s_t, a_t)$
- It describes a problem, not a solution
- Markov property : $p(s^{t+1}|s^t, a^t) = p(s^{t+1}|s^t, a^t, s^{t-1}, a^{t-1}, ...s^0, a^0)$
- Reactive agents $a_{t+1} = f(s_t)$, without internal states nor memory
- In an MDP, a memory of the past does not provide any useful advantage

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

Reinforcement learning

- In Dynamic Programming (planning), T and r are given
- Reinforcement learning goal: build π^* without knowing T and r
- Model-free approach: build π^* without estimating T nor r
- Actor-critic approach: special case of model-free
- Model-based approach: build a model of T and r and use it to improve the policy

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

Incremental estimation

- Estimating the average immediate (stochastic) reward in a state s
- $E_k(s) = (r_1 + r_2 + ... + r_k)/k$
- $E_{k+1}(s) = (r_1 + r_2 + ... + r_k + r_{k+1})/(k+1)$
- Thus $E_{k+1}(s) = k/(k+1)E_k(s) + r_{k+1}/(k+1)$
- Or $E_{k+1}(s) = (k+1)/(k+1)E_k(s) - E_k(s)/(k+1) + r_{k+1}/(k+1)$
- Or $E_{k+1}(s) = E_k(s) + 1/(k+1)[r_{k+1} - E_k(s)]$
- Still needs to store k
- Can be approximated as

$$E_{k+1}(s) = E_k(s) + \alpha[r_{k+1} - E_k(s)] \tag{1}$$

- Converges to the true average (slower or faster depending on α) without storing anything
- Equation (1) is everywhere in reinforcement learning

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 $V(s_t) = r_{t+1} + \gamma V(s_{t+1})$
- $\delta_k = r_{k+1} + \gamma V(s_{k+1}) - V(s_k)$: Reward Prediction Error (RPE)
- Measures the error between current and expected values of V
- TD learning: If δ positive, increase V, if negative, decrease V
- $V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$

TD learning: limitation

- TD(0) evaluates $V(s)$
- One cannot infer $\pi(s)$ from $V(s)$ without knowing T: one must know which a leads to the best $V(s')$
- Three solutions:
  - Work with $Q(s, a)$ rather than $V(s)$ (Sarsa and *Q-Learning*)
  - Learn a model of T: model-based (or indirect) reinforcement learning
  - Actor-critic methods (simultaneously learn V and update π)

Sarsa

- Reminder (TD): $V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$
- Sarsa: For each observed $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$:
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
- Policy: perform exploration (e.g. ϵ-*greedy*)
- One must know the action a_{t+1}, thus constrains exploration
- On-policy method: more complex convergence proof

Singh, S. P., Jaakkola, T., Littman, M. L., & Szepesvari, C. (2000). Convergence Results for Single-Step On-Policy Reinforcement Learning Algorithms. *Machine Learning*, 38(3):287–308.

Q-Learning

- For each observed $(s_t, a_t, r_{t+1}, s_{t+1})$:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- $\max_{a \in A} Q(s_{t+1}, a)$ instead of $Q(s_{t+1}, a_{t+1})$
- Off-policy method: no more need to know a_{t+1}
- Policy: perform exploration (e.g. ϵ-*greedy*)
- Convergence proved given infinite exploration [Dayan & Sejnowski, 1994]

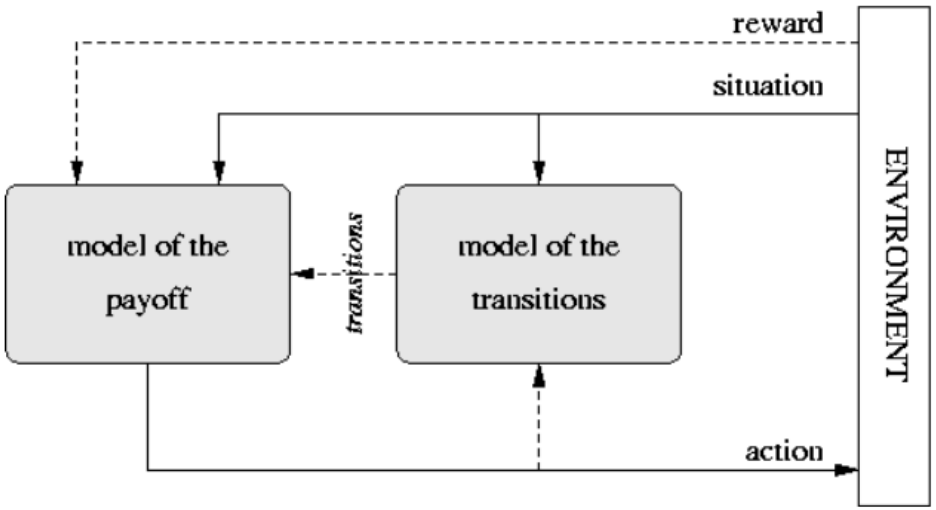
Watkins, C. J. C. H. (1989). *Learning with Delayed Rewards*. PhD thesis, Psychology Department, University of Cambridge, England.

Q-Learning in practice

(Q-learning: the movie)

- Build a states×actions table (*Q-Table*, eventually incremental)
- Initialise it (randomly or with 0 is not a good choice)
- Apply update equation after each action
- Problem: it is (very) slow

Model-based Reinforcement Learning



- General idea: planning with a learnt model of T and r is performing back-ups "in the agent's head" ([Sutton, 1990, Sutton, 1991])
- Learning T and r is an incremental self-supervised learning problem
- Several approaches:
  - Draw random transition in the model and apply TD back-ups
  - Dyna-PI, Dyna-Q, Dyna-AC
  - Better propagation: Prioritized Sweeping

Moore, A. W. & Atkeson, C. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130.

Dyna architecture and generalization

(Dyna-like video (good model))
(Dyna-like video (bad model))

- Thanks to the model of transitions, Dyna can propagate values more often
- Problem: in the stochastic case, the model of transitions is in $card(S) \times card(S) \times card(A)$
- Usefulness of compact models
- MACS: Dyna with generalisation (Learning Classifier Systems)
- SPITI: Dyna with generalisation (Factored MDPs)

Gérard, P., Meyer, J.-A., & Sigaud, O. (2005) Combining latent learning with dynamic programming in MACS. *European Journal of Operational Research*, 160:614–637.

Degris, T., Sigaud, O., & Wuillemin, P.-H. (2006) Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems. *Proceedings of the 23rd International Conference on Machine Learning (ICML'2006)*, pages 257–264

From *Q-Learning* to Actor-Critic (1)

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

- In $Q - learning$, given a $Q - Table$, one must determine the max at each step
- This becomes expensive if there are numerous actions (optimization in continuous action case)

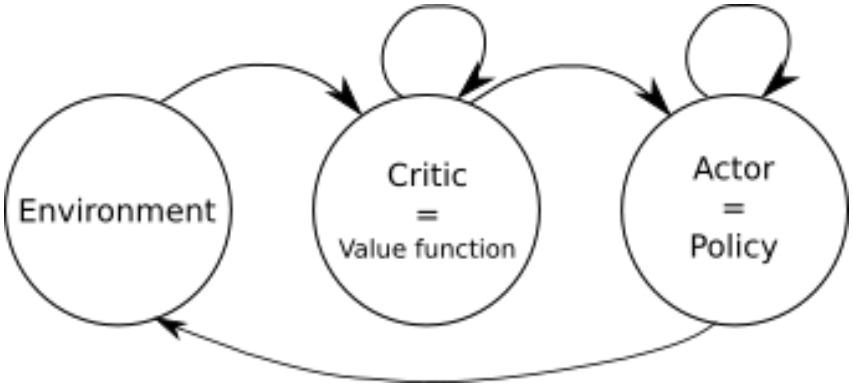
From *Q-Learning* to Actor-Critic (2)

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

state	chosen action
e_0	a_1
e_1	a_2
e_2	a_2
e_3	a_2
e_4	a_1
e_5	a_1

- One can store the best value for each state
- Storing the max is equivalent to storing the policy
- Update the policy as a function of value updates (only look for the max when decreasing max action)
- Note: looks for local optima, not global ones anymore

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)

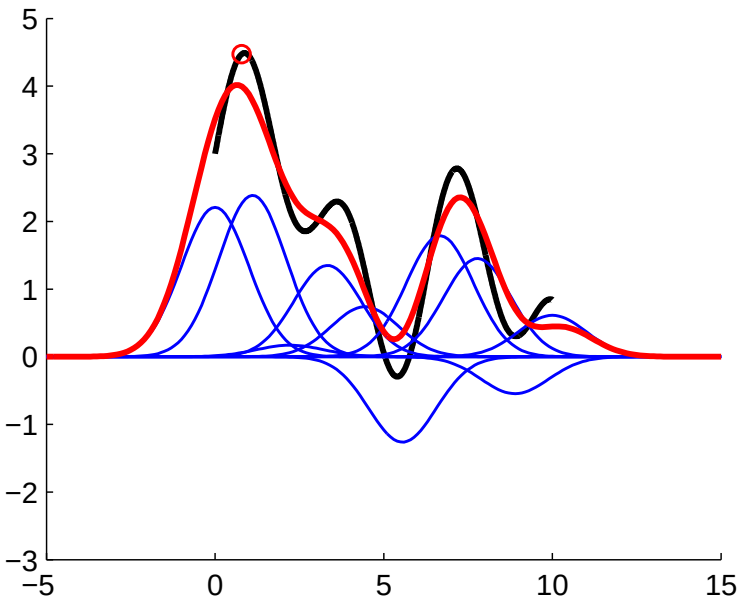
A few messages

- Dynamic programming and reinforcement learning methods can be split into pure actor, pure critic and actor-critic methods
- Dynamic programming, value iteration, policy iteration are when you know the transition and reward functions
- Actor critic RL is a model-free, PI-like algorithm
- Model-based RL combines dynamic programming and model learning

Questions

- SARSA is on-policy and Q-learning is off-policy
  Right or Wrong ?
- The actor-critic approach is model-based
  Right or Wrong ?
- In SARSA, the policy is represented implicitly through the critic
  Right or Wrong ?

Parametrized representations



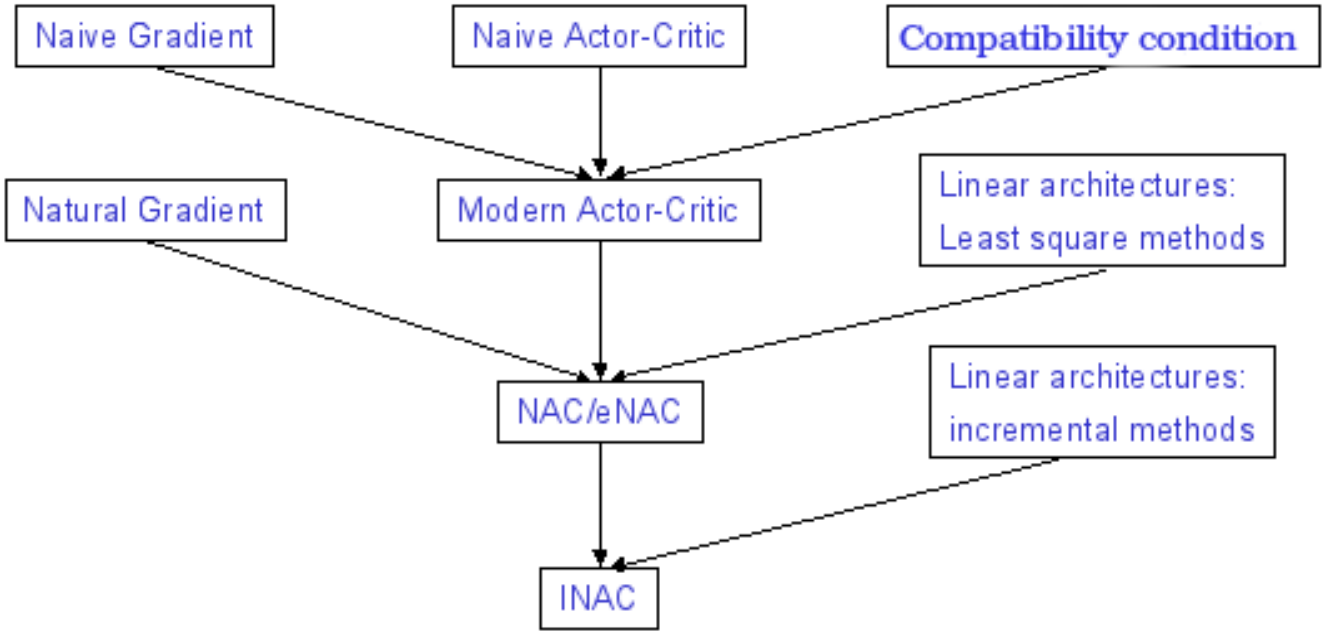
- To represent a continuous function, use features and a vector of weights (parameters)
- Learning tunes the weights

- Linear architecture: linear combination of features
- A deep neural network is not a linear architectures: weights also "inside" the features
- Two parametrized representations:
  - In policy gradient methods: of the policy $\pi_{\mathbf{w}}(a_t|s_t)$
  - In actor-critic methods: and also of the critic $Q(s_t, a_t|\theta)$

Optimization over continuous actions

- In RL, you need a max over actions
- If the action space is continuous, this is a difficult optimization problem
- Policy gradient methods and actor-critic methods mitigate the problem by looking for a local optimum (Pontryagine methods vs Bellman methods)

Quick history of previous attempts (J. Peters' and Sutton's groups)



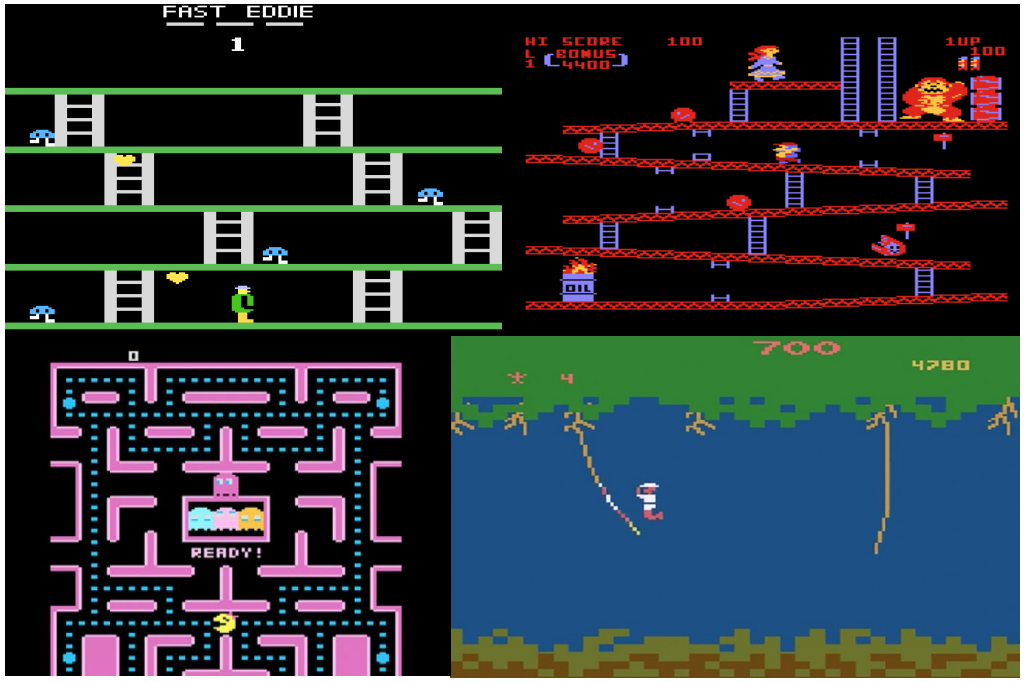
- Those methods proved inefficient for robot RL
- Keys issues: value function estimation based on linear regression is too inaccurate, tuning the stepsize is critical

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.

General motivations for Deep RL

- Approximation with deep networks provided enough computational power can be very accurate
- Discover the adequate features of the state in a large observation space
- All the processes rely on efficient backpropagation in deep networks
- Available in CPU/GPU libraries: TensorFlow, theano, caffe, Torch... (RProp, RMSProp, Adagrad, Adam...)

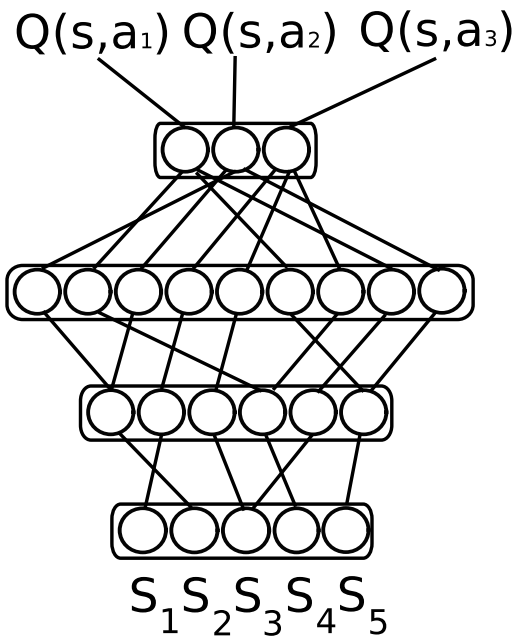
DQN: the breakthrough



- DQN: Atari domain, Nature paper, small discrete actions set
- Learned very different representations with the same tuning

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.

The Q-network in DQN



- Limitation: requires one output neuron per action
- Select action by finding the max (as in *Q-Learning*)
- Q-network parameterized by θ

Learning the Q-function

- Supervised learning: minimize a loss-function, often the squared error w.r.t. the output:

$$L(s,a) = (y^*(s,a) - Q(s,a|\theta))^2 \tag{2}$$

  by backprop on critic weights θ
- For each sample i, the Q-network should minimize the RPE:

$$\delta_i = r_i + \gamma \max_a Q(s_{i+1}, a|\theta) - Q(s_i, a_i|\theta)$$

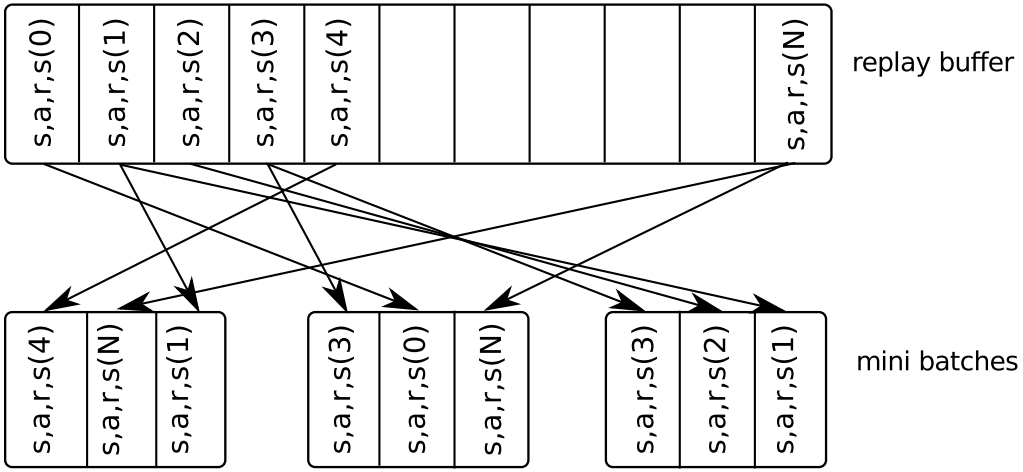
- Thus, given a minibatch of N samples $\{s_i, a_i, r_i, s_{i+1}\}$, compute $y_i = r_i + \gamma \max_a Q(s_{i+1}, a|\theta')$
- So update θ by minimizing the loss function

$$L = 1/N \sum_i (y_i - Q(s_i, a_i|\theta))^2 \tag{3}$$

Trick 1: Stable Target Q-function

- $y_i = r_i + \gamma \max_a Q(s_{i+1}, a)|\theta)$ is a function of Q
- Thus this is not truly supervised learning, and this is unstable
- Idea: compute the critic loss function from a separate target network $Q'(...|\theta')$
- So rather compute $y_i = r_i + \gamma \max_a Q'(s_{i+1}, a|\theta')$
- θ is updated only each K iterations (so "periods of supervised learning")

Trick 2: Sample buffer



- 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
- Use training minibatches, to make profit of GPU
- The replay buffer management policy is an issue

de Bruin, T., Kober, J., Tuyls, K., & Babuška, R. (2015) The importance of experience replay database composition in deep reinforcement learning. In *Deep RL workshop at NIPS 2015*

Double-DQN

- The max operator in the RPE results in the propagation of over-estimation
- This max operator is used both for action choice and value propagation
- Double *Q-Learning*: separate both calculations (Van Hasselt)
- Double-DQN: make profit of the target network: propagate on target network, select max on Q-network,
- Minor change with respect to DQN
- But with a much better performance
- Recent paper on double SARSA

Van Hasselt, H., Guez, A., & Silver, D. (2015) Deep reinforcement learning with double q-learning. *CoRR, abs/1509.06461*

Ganger, M., Duryea, E., & Hu, W. (2016) Double sarsa and double expected sarsa with shallow and deep learning. *Journal of Data Analysis and Information Processing*, 4(04):159-176

Prioritized Experience Replay

- Samples with a greater TD error have a higher probability of being selected
- Favors the replay of new (s, a) pairs (largest TD error), as in $R - max$
- Several minor hacks, and interesting discussion
- Converges twice faster

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015) Prioritized experience replay. *arXiv preprint arXiv:1511.05952*

- Other state-of-the-art methods: Gorilla, A3C: parallel implementations without replay buffers

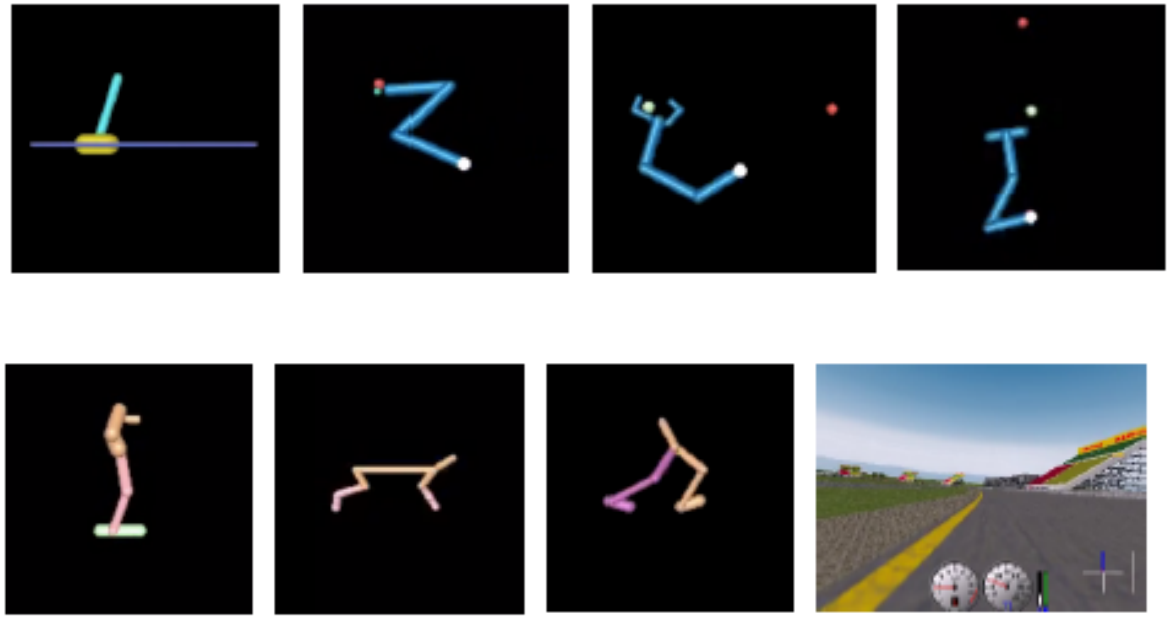
Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016) Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*

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 $>$ 280 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*.

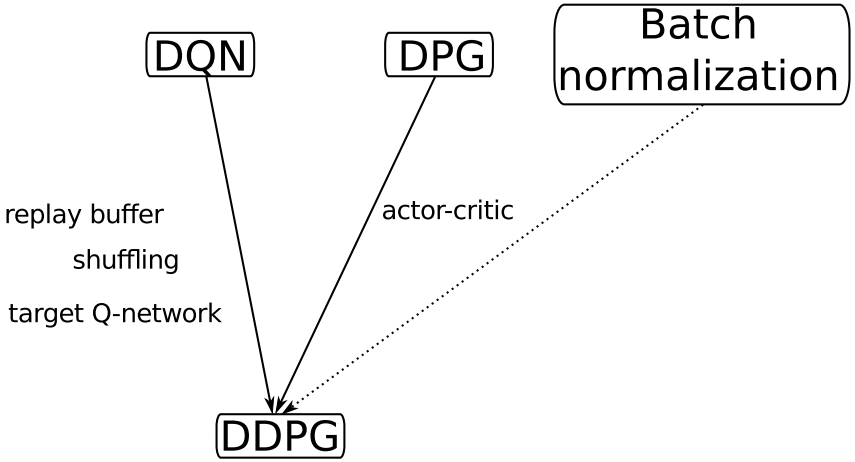
Applications: impressive results



- End-to-end policies (from pixels to control)
- Works impressively well on "More than 20" (27-32) such domains
- Some domains coded with MuJoCo (Todorov) / TORCS
- OpenAI gym gives access to those benchmarks

Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2016) Benchmarking deep reinforcement learning for continuous control. *arXiv preprint arXiv:1604.06778*.

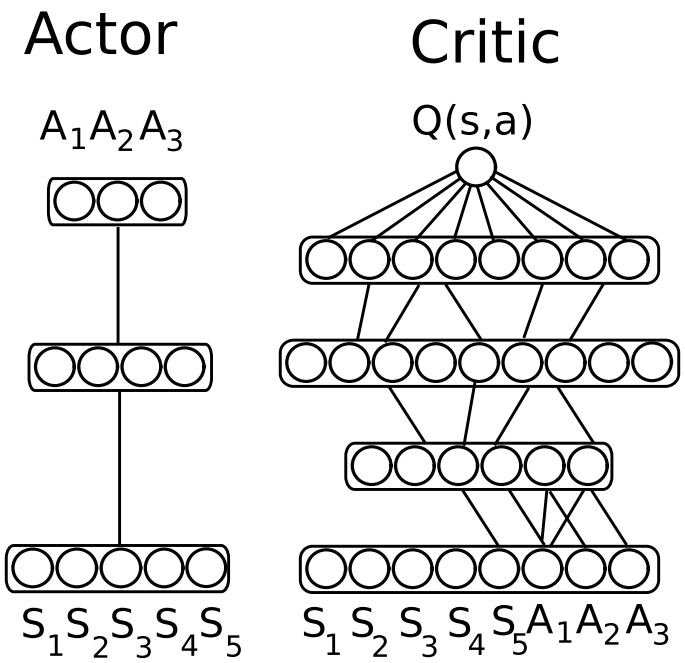
DDPG: ancestors



- Most of the actor-critic theory for continuous problem is for stochastic policies (policy gradient theorem, compatible features, etc.)
- DPG: an efficient gradient computation for deterministic policies, with proof of convergence

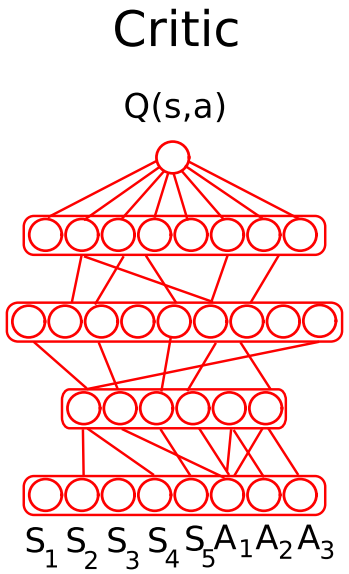
Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014) Deterministic policy gradient algorithms. In *ICML*.

General architecture



- Any neural network structure
- Actor parametrized by **w**, critic by θ
- All updates based on backprop

Training the critic



- Same idea as in DQN, but with an actor-critic update rather than *Q-Learning*
- Minimize the RPE: $\delta_t = r_t + \gamma Q(s_{t+1}, \pi(s_t)|\theta) - Q(s_t, a_t|\theta)$
- Given a minibatch of N samples $\{s_i, a_i, r_i, s_{i+1}\}$ and a target network Q', compute $y_i = r_i + \gamma Q'(s_{i+1}, \pi(s_{i+1})|\theta')$
- And update θ by minimizing the loss function

$$L = 1/N \sum_i (y_i - Q(s_i, a_i|\theta))^2 \quad (4)$$

From DQN: Target network

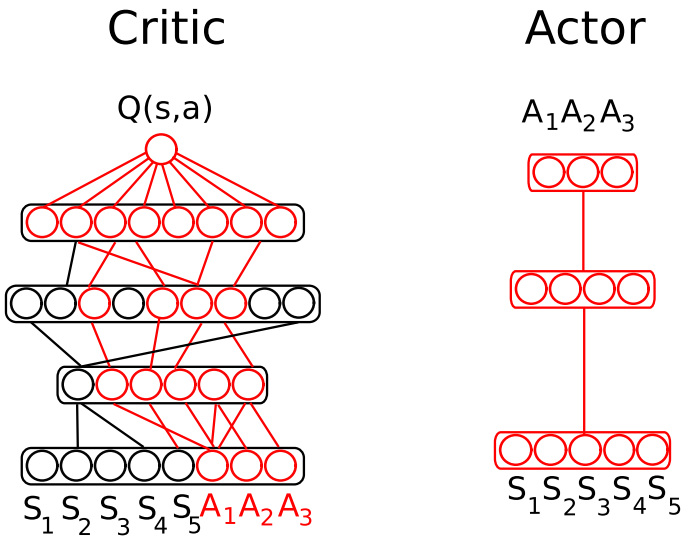
- In DDPG, instead of scarce updates, slow evolution of Q' and π'

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

- The same applies to μ, μ' (slow evolution of the actor)
- From the empirical study, this is a critical trick
- NB: actor-critic tuning is known to be tedious!

Pfau, D. & Vinyals, O. (2016). Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945*

Training the actor



- Deterministic policy gradient theorem: the true policy gradient is

$$\nabla_{\mathbf{w}}\pi(s,a) = \mathbb{E}_{\rho(s)}[\nabla_a Q(s,a|\theta)\nabla_{\mathbf{w}}\pi(s|\mathbf{w})] \tag{5}$$

- $\nabla_a Q(s,a|\theta)$ is obtained by computing the gradient over actions of $Q(s,a|\theta)$
- Gradient over actions $\sim$ gradient over weights (symmetric roles of weights and inputs)
- $\nabla_a Q(s,a|\theta)$ is used as backprop error signal to update the actor weights.
- Comes from NFQCA

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

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 (4) (alternative: do it after 4?)
4. Compute $\nabla_a Q(s, a|\theta)$
5. Update the actor, using (5)

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 θ' and μ' updates (instead of copying as in DQN)

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 3: Batch Normalization

- 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.*

Back to natural gradient: other ideas

- Using the advantage function leads to natural gradient (vs vanilla gradient)
- Batch normalization and Weight normalization are specific reparametrization methods
- Computing the natural gradient is also a reparametrization method
- Natural Neural networks define a reparametrization that compute the natural gradient (to be investigated)

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

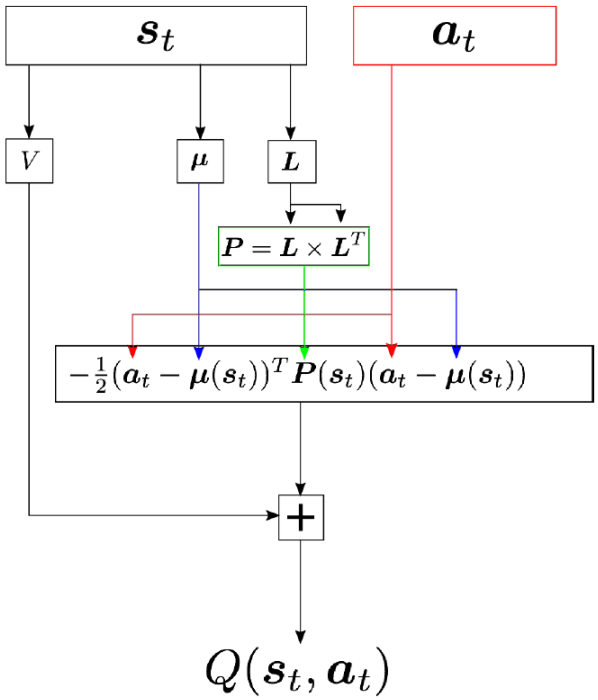
Desjardins, G., Simonyan, K., Pascanu, R., et al. (2015) Natural neural networks, In *Advances in Neural Information Processing Systems* (pp. 2062–2070).

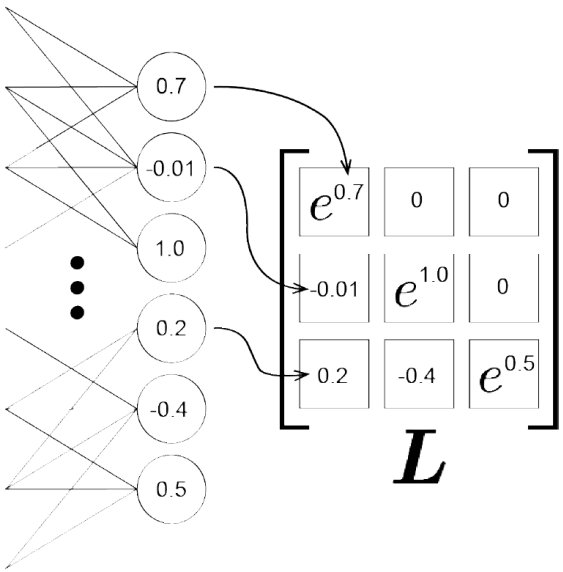
NAF: Approximate the advantage function

- Reminder: in *Q-Learning*, high cost to select best action
- Here, set a specific form to Q-network so as to find the best action easily
- Advantage function: $A(s_i, a_i|\theta) = Q(s_i, a_i|\theta) - max_a Q(s_i, a|\theta)$
- $V(s_i) = max_a Q(s_i, a|\theta)$
- $Q(s_i, a_i|\theta^Q) = A(s_i, a_i|\theta^A) + V(s_i|\theta^V)$
- $A_\theta(s_i, a_i|\theta^A) = -\frac{1}{2}(a_i - \mu(s_i|\theta^\mu))^T P(s_i|\theta^P)(a_i - \mu(s_i|\theta^\mu))$

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

NAF: the network





- All neural nets are $dim(s) \times dim(a)$
- Implemented with 2 layers of 200 relu units
- The μ network is the actor
- Outperforms DDPG on some benchmarks
- Other tricks in the paper: use iLQG for model-based acceleration

Status

- DDPG used successfully on continuous Mountain Car: much more data efficient than CMA-ES
- I failed to tune it for a 4D/6D motor control problem with noisy perception and delays
- NAF is used in real robotics settings with some success
- Now working accurately on the stability issue

Gu, S., Holly, E., Lillicrap, T., & Levine, S. (2016a) Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1610.00633*

TRPO, PPO

- Theory: monotonous improvement w.r.t. the cost function
- Practice: good grip on the step size
- Follows the natural gradient
- More stable, performs well in practice

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015) Trust region policy optimization. *CoRR, abs/1502.05477*

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015b) High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., & Silver, D. (2016) Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

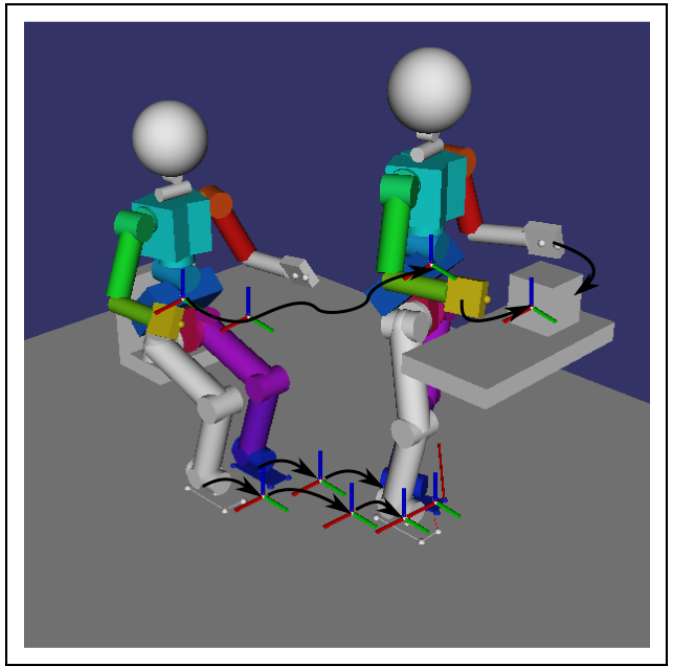
The frontiers

- Two more recent papers: ACER and Q-Prop
- Confirm that DDPG is tricky to tune
- Combine the TRPO and DDPG approaches to get more efficient and more stable
- It gets really complicated
- The fundamental instability issue is not solved
- One cannot compete with OpenAI, Google US and Google DeepMind on this topic...

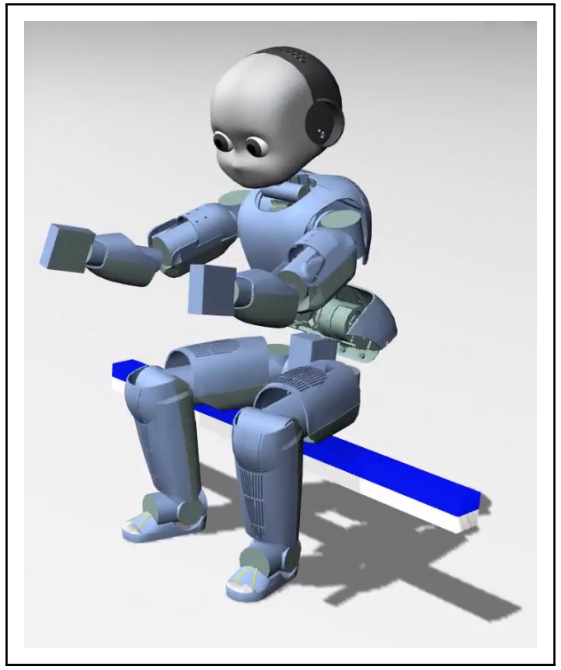
Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016). Sample efficient actor-critic with experience replay.arXiv preprint arXiv:1611.01224.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2016) Q-prop: Sample-efficient policy gradient with an off-policy critic.arXiv preprint arXiv:1611.02247.

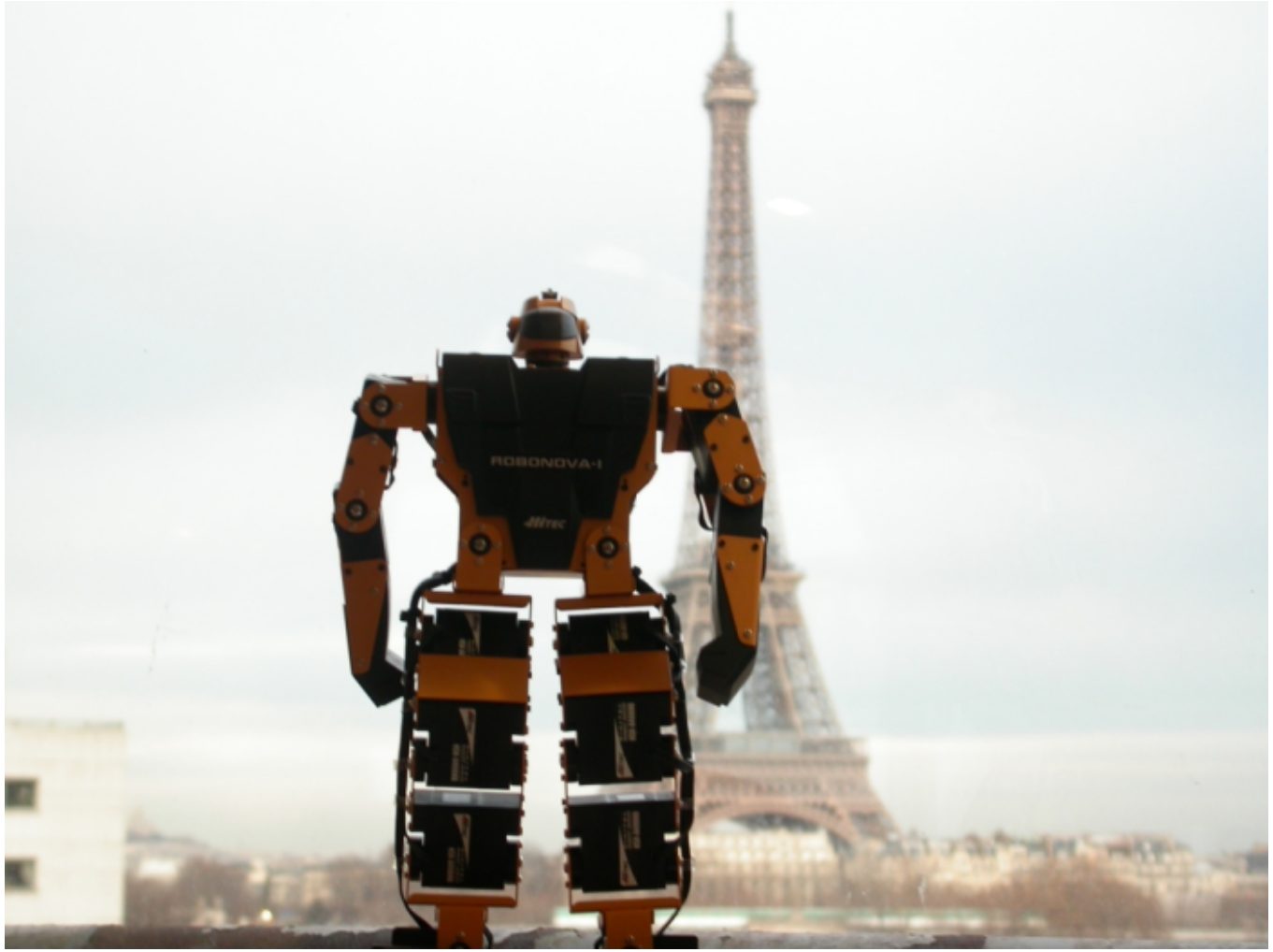
Reinforcement learning for robots (old)



Reinforcement learning for robots (new)



Any question?



Dayan, P. & Sejnowski, T. (1994).
TD(lambda) converges with probability 1.
Machine Learning, 14(3), 295–301.

de Bruin, T., Kober, J., Tuyls, K., & Babuška, R. (2015).
The importance of experience replay database composition in deep reinforcement learning.
In *Deep RL workshop at NIPS 2015*.

Degris, T., Sigaud, O., & Wuillemin, P.-H. (2006).
Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems.
In *Proceedings of the 23rd International Conference on Machine Learning* (pp. 257–264). CMU, Pennsylvania.

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

Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2016).
Benchmarking deep reinforcement learning for continuous control.
arXiv preprint arXiv:1604.06778.

Ganger, M., Duryea, E., & Hu, W. (2016).
Double sarsa and double expected sarsa with shallow and deep learning.
Journal of Data Analysis and Information Processing, 4(04), 159–176.

Gérard, P., Meyer, J.-A., & Sigaud, O. (2005).
Combining latent learning with dynamic programming in MACS.
European Journal of Operational Research, 160, 614–637.

Gu, S., Holly, E., Lillicrap, T., & Levine, S. (2016a).
Deep reinforcement learning for robotic manipulation.
arXiv preprint arXiv:1610.00633.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., & Levine, S. (2016b).

Q-prop: Sample-efficient policy gradient with an off-policy critic.
arXiv preprint arXiv:1611.02247.

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

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., & Silver, D. (2016).
Learning and transfer of modulated locomotor controllers.
arXiv preprint arXiv:1610.05182.

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., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016).
Asynchronous methods for deep reinforcement learning.
arXiv preprint arXiv:1602.01783.

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.

Moore, A. W. & Atkeson, C. (1993).
Prioritized sweeping: Reinforcement learning with less data and less real time.

Machine Learning, 13, 103–130.

Pfau, D. & Vinyals, O. (2016).
Connecting generative adversarial networks and actor-critic methods.
arXiv preprint arXiv:1610.01945.

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

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015).
Prioritized experience replay.
arXiv preprint arXiv:1511.05952.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015a).
Trust region policy optimization.
CoRR, abs/1502.05477.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., & Abbeel, P. (2015b).
High-dimensional continuous control using generalized advantage estimation.
arXiv preprint arXiv:1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017).
Proximal policy optimization algorithms.
arXiv preprint arXiv:1707.06347.

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.*

Singh, S. P., Jaakkola, T., Littman, M. L., & Szepesvari, C. (2000).
Convergence Results for Single-Step On-Policy Reinforcement Learning Algorithms.
Machine Learning, 38(3), 287–308.

Sutton, R. S. (1990).
Integrating architectures for learning, planning, and reacting based on approximating dynamic programming.
In *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216–224). San Mateo, CA: Morgan Kaufmann.

Sutton, R. S. (1991).
DYNA, an integrated architecture for learning, planning and reacting.
SIGART Bulletin, 2, 160–163.

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.

Van Hasselt, H., Guez, A., & Silver, D. (2015).
Deep reinforcement learning with double q-learning.
CoRR, abs/1509.06461.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., & de Freitas, N. (2016).
Sample efficient actor-critic with experience replay.
arXiv preprint arXiv:1611.01224.

Watkins, C. J. C. H. (1989).
Learning with Delayed Rewards.
PhD thesis, Psychology Department, University of Cambridge, England.