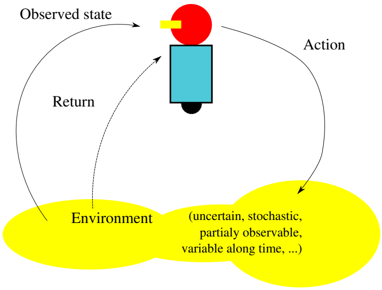
Reinforcement learning (and bandits)

Philippe Preux
philippe.preux@inria.fr

SequeL, Inria Lille, CRIStAL, Univ. Lille

Sequential Decision Making under Uncertainty

Reinforcement learning: Learning by trial-and-error

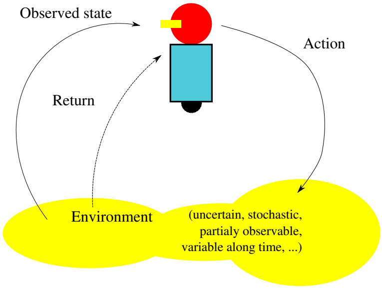


Goal: learn to map state to the optimal action to perform in ths state.

- Markov dynamics
- transition function unknown
- reward function unknown
- if both are known: planning problem: solved by dynamic programming

Sequential Decision Making under Uncertainty

Reinforcement learning: Learning by trial-and-error



Goal: learn to map state to the optimal action to perform in ths state.

Two main approaches:

- estimate the value of states; Bellman equation; TD-learning; Q-Learning; approximate dynamic programming
- estimate the optimal policy; Pontryagin's approach; REINFORCE; policy gradient
- + their combination: actor/critic approach

May also involve model learning

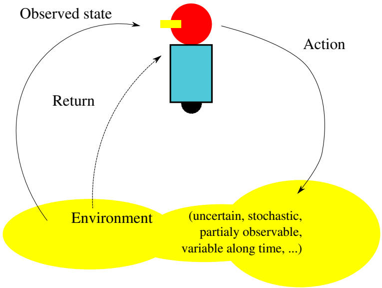
Reinforcement learning is not deep learning

Reinforcement learning is not neural nets

We may use neural net (shallow or deep) in reinforcement learning implementation.

Reinforcement learning

Anything deep in there?



Learns to map state to the optimal action to perform in ths state.

- small state space: tabular algorithm
  no generalization.
- otherwise: use a function approximator = a data structure / a function that inputs a state and output an action.
  A multi-layer perceptron is one kind of function approximator.

Application to games

- Backgammon: 1994: TD-Gammon
  a 1 hidden layer with 192 units. Hand-crafted input features.
  1.5 10^6 self-plays
  expert level
  Backgammon is non deterministic. Branching factor ≈ 800, horizon is potentially ∞.
- Alpha Go: 2015: trained with human plays, first go program to beat a human expert. End-to-end.
- Alpha Zero: 2017: trained by self-play using only the rules of the game: 40-80 layers.
  Go is deterministic. Branching factor < 400, horizon < 400.
- Alpha Zero trained by self-play on other board games (chess, draughts, othello, ...).
- anyway, this all remains ridiculously small problems.

Some remarks about RL

- RL requires lots of interaction agent-environment to learn
- better if input features are worked on.

- RL also requires lots of parameter tuning, ...
- modern RL (alpha zero) involves lots of tricks to make learning faster

Computation burden

- simulation of the environment
- action selection (*e.g.* Monte Carlo tree search in discrete RL; or a continuous optimization problem in contnuous RL)
- the DL part
- the design
- the tuning
- the evaluation

We never really know whether what we try is not working because it can't, or because we didn't find a way to make it work, or we did not run the learner long enough (or there are bugs).

Bandits

Bandits = RL with no state $\rightsquigarrow$ action selection.

Find the most rewarding arm/action/choice among K alternatives.

A lot of theory. Theory does help. Algorithms with optimal performance bounds.
Takes into account structure of the problem, that is relations between arms: arm features, graph of bandits, metric space of bandits, ...

Applications in HPC

- scheduling
- placement
- check pointing
- ...