From MCTS to Hilbert

Olivier Sigaud

Sorbonne Université
http://www.isir.upmc.fr/personnel/sigaud

Motivation

- A domain independent algorithm to plan in one's head to determine the best next action
- Example: two-player games (Chess, Go...).
- Minimax defeated Kasparov in 1998, was considering the whole tree, too expensive at Go
- MCTS was the leading technique at Go before AlphaZero
- Requires an internal simulator
- Requires a capability to reset anywhere
- Very efficient tree search method

Gelly, S., Wang, Y., Munos, R., and Teytaud, O. Modification of UCT with patterns in Monte-Carlo go. Technical Report 32, RR-6062, INRIA, 2006.

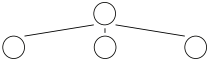
Overview

- Somewhere between breadth-first and depth-first search
- Similar to A^* without the admissible heuristic
- The cost is in the numerous simulations $\rightarrow$ AlphaZero improves this
- Four processes: Selection, Expansion, Simulation, Update

Initial step

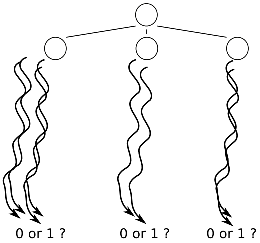
- A node represents a discrete state, an edge represents a discrete action
- The process starts with an empty node
- This node corresponds to the current state where the next action has to be chosen

Initial step: Expansion



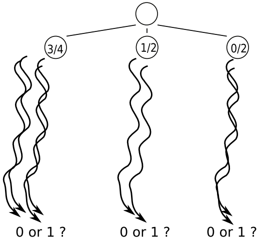
- The MCTS agent tries actions (in its head), resulting in adding child nodes

Initial step: Simulation



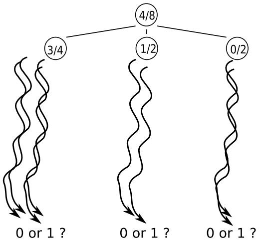
- From each selected node, it performs random simulations (Monte Carlo) to evaluate the node (without adding nodes yet)
- Initial child node selection is random

Initial step: Update



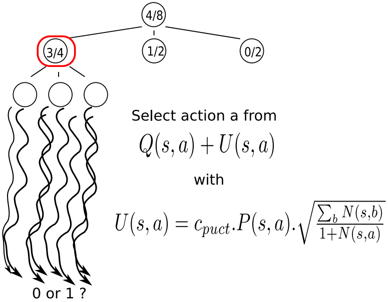
- It updates the values of children based on the statistics of the simulations
- The value is a state value $V(s)$

Initial step: Update parents



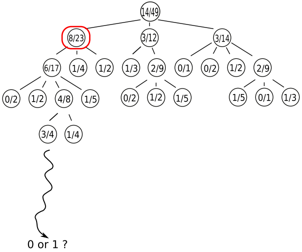
- It also updates the values of parents
- Note that state values $V(s)$ could be changed into state-action values $Q(s, a)$ using $Q(s, a) = r(s, a) + \gamma V(s')$
- In Go, $Q(s, a) = V(s')$ (no intermediate reward)

Selection



- Selection operates over all expanded nodes
- That's where the reset-anywhere property is necessary
- It favors leaf nodes with a higher chance of success
- But it avoids ignoring too much lower success nodes
- A lower $N(s,a)$ results in a higher $U(s,a)$
- The selected node is expanded, and the process is repeated

Action Selection

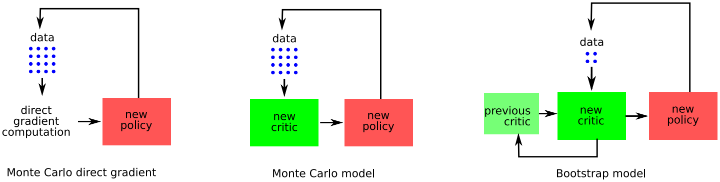


- After some budget, the search process stops
- The agent performs the action leading to the most visited first level child
- In exploration mode, some noise is added
- The current agent state is updated, and the process starts again
- MPC-like process
- A lot of computations are forgotten...

Summary

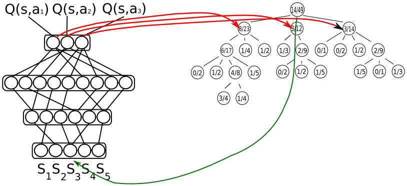
- MCTS plays well Go or chess, but is quite inefficient
- Areas for improvement:
  - Avoid forgetting $Q(s,a)$ after each step
  - Avoid using Monte Carlo simulations again each time to evaluate $Q(s,a)$
  - Instead of running random simulations, play good moves with higher probabilities
- Adding a critic network solves these issues

Actor-Critic vs Monte Carlo



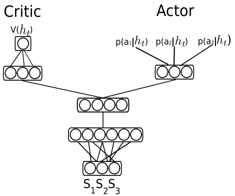
- Monte Carlo direct gradient: Estimate $Q(s,a)$ over rollouts
- Monte Carlo model: learn a model $\hat{Q}(s,a)$ over rollouts using MC regression, throw it away after each update
- Bootstrap: Update a model $\hat{Q}(s,a)$ over samples using TD methods, keep it over policy gradient steps
- The bootstrap approach is much more sample efficient
- It introduces bias and reduces variance

MCTS + Critic



- Learns a critic $\hat{Q}(s,a)$ for all states over all rollouts
- Using a DQN-like architecture
- Still builds a plan with an MPC-like approach, not using $\max_a \hat{Q}(s,a)$ as policy
- The MCTS search process helps balancing samples, favors exploration
- In AlphaZero:
  - Instead of playing random rollouts, can play rollouts driven by $\hat{Q}(s,a)$
  - The critic $\hat{Q}(s,a)$ can be pre-trained with expert moves (AlphaGo vs AlphaZero)

AlphaZero: from DQN-like to actor-critic

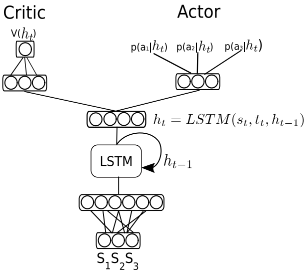


- Learning a policy and a $\hat{V}(s)$ function is more efficient than using a $\hat{Q}(s,a)$ function

Motivation

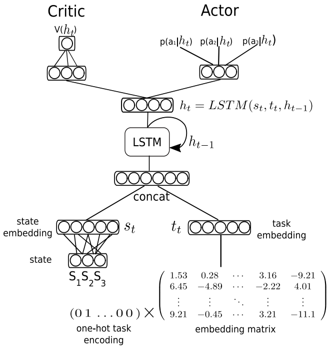
- AlphaZero is very efficient at solving single-task, discrete action problems
- AlphaNPI is an extension to multitask, hierarchical problem solving

Step 1: dealing with non-Markov problems



- An LSTM stores some context from the previous state

Step 2: making it multitask



- Using the task as input makes the architecture multitask (equivalent to GC-RL)
- Additional feature inherited from NPI: using state and task embedding
- Impact of embeddings not studied, ablation needed

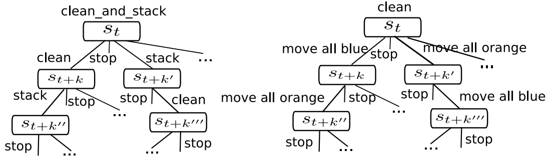
Step 3: defining a (loose) hierarchy of tasks

program	description	level
BUBBLESORT	sort the list	3
RESET	move both pointers to the extreme left of the list	2
Bubble	make one pass through the list	2
RSHIFT	move both pointers once to the right	1
LSHIFT	move both pointers once to the left	1
COMPSWAP	if both pointers are at the same position, move pointer 2 to the left, then swap elements at pointers positions if left element > right element	1
PTR_2_L	move pointer 2 to the left	0
PTR_1_L	move pointer 1 to the left	0
PTR_1_R	move pointer 1 to the right	0
PTR_2_R	move pointer 2 to the right	0
SWAP	swap elements at the pointers positions	0
STOP	terminates current program	0

Table 4: Program library for the list sorting environment.

- The list of task is where expert knowledge is inserted
- A task can only call a subtask of lower or equivalent level
- This helps constraining recursive tree search
- Additional constrainsts with preconditions can be used

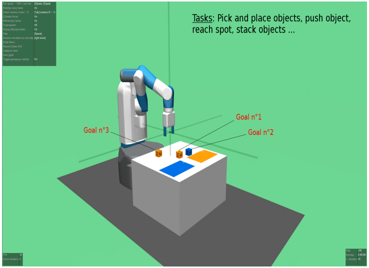
Recursive tree search: implementation



(a) (b)

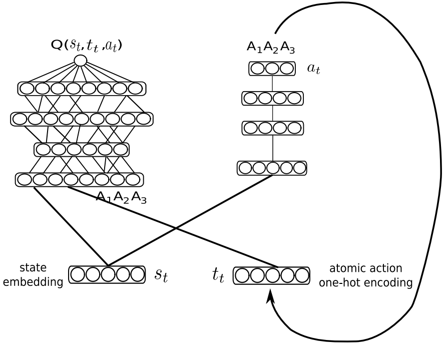
- When a task calls a subtask
  - A subtree is created
  - The current task context is stored into a stack
  - And unstacked upon termination (as when calling a function in programming languages)
- Thus in AlphaNPI we have a tree of MCTS searches (tree of trees)

Summary



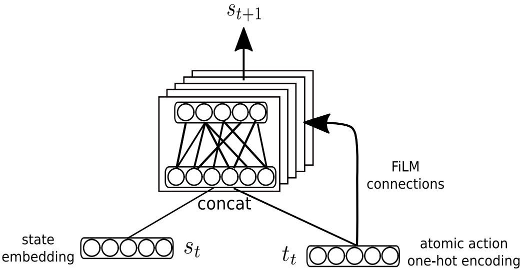
- AlphaNPI is applied to discrete actions
- HILBERT deals with continuous actions
- It learns forward models of the lowest level
- It provides a very sample efficient approach to continuous action HRL

Low-level controller: GC-RL



- GC-RL using DDPG (or SAC) + HER

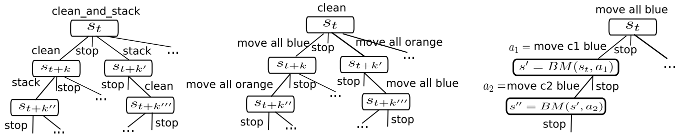
Learning a behavioral model of low level controller



- This is a supervised learning problem
- The FiLM layer improves accuracy

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018) Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32

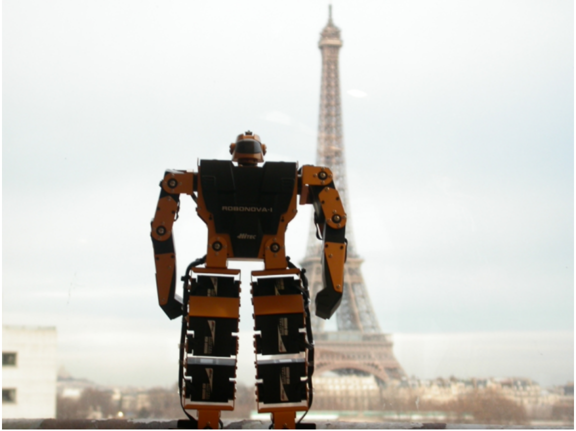
Recursive tree search: continuous action case



(a) (b) (c)

- The lowest level stops the recursion
- HILBERT can perform hierarchical planning without rolling the low-level policy
- By using the behavioral model, higher level planning is learned without sampling
- Extremely sample efficient search approach

Any question?



Send mail to: `Olivier.Sigaud@upmc.fr`

Gelly, S., Wang, Y., Munos, R., and Teytaud, O.
Modification of UCT with patterns in monte-carlo go.
Technical Report 32, RR-6062, INRIA, 2006.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A.
Film: Visual reasoning with a general conditioning layer.
In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.