Monte Carlo Tree Search (MCTS) algorithms

Vincent François-Lavet

Foreword

In this lesson, we will cover the **Monte Carlo Tree Search (MCTS)** algorithms.

- The context (and notations) is recalled/introduced in the first few slides.

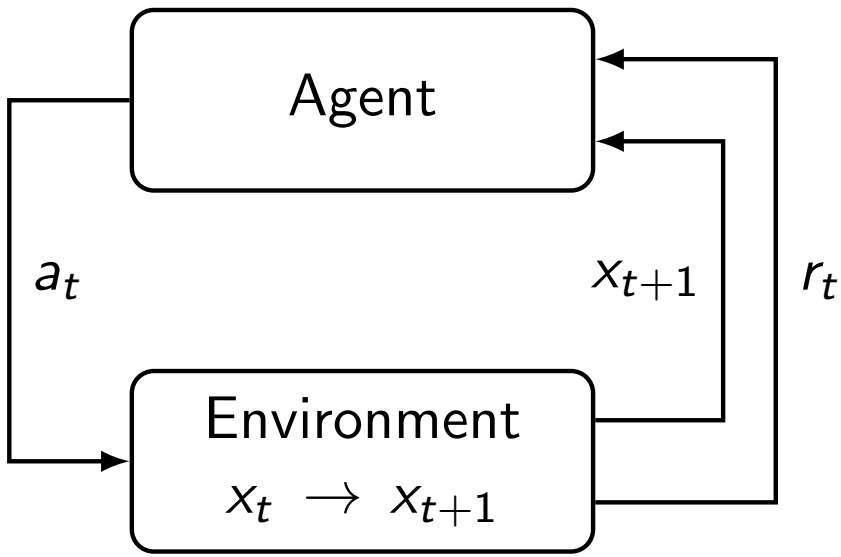
Please do not hesitate to ask questions.

Outline

Recall on the MDP setting

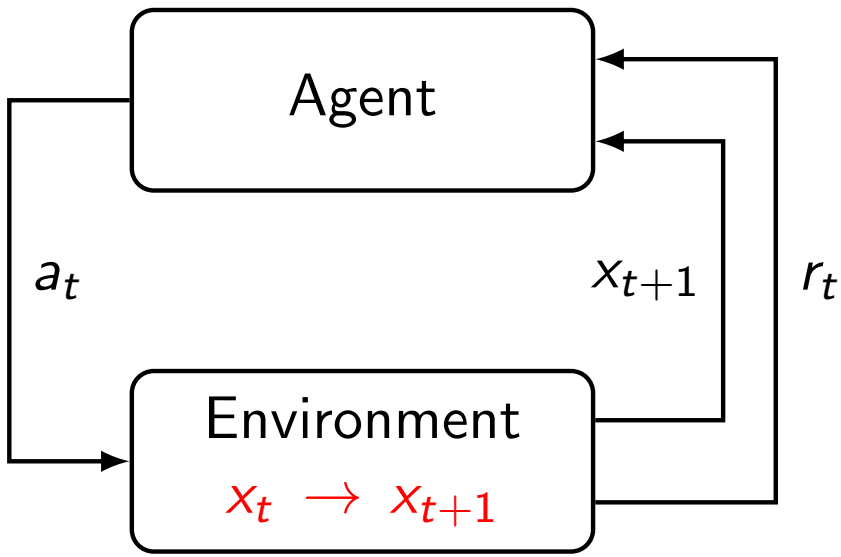
Objective

From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Objective

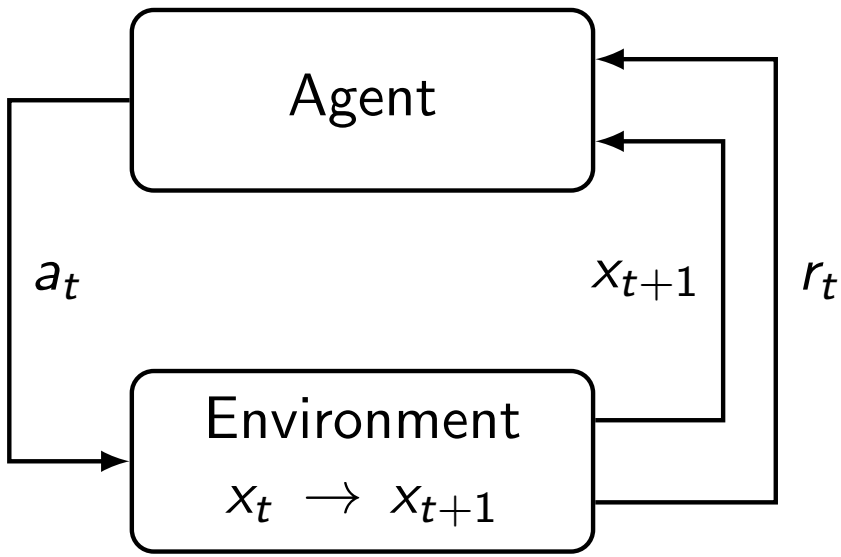
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



transitions
are usually
stochastic

Objective

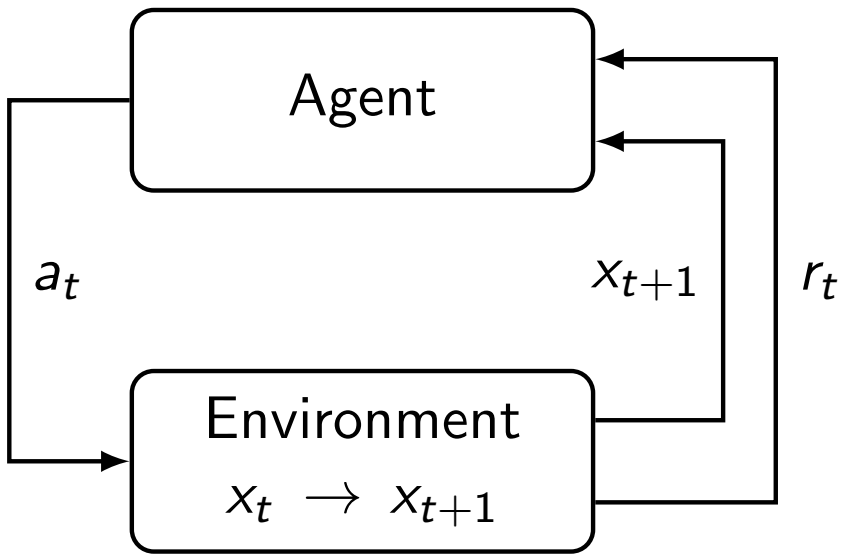
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Observations and
actions may be
high dimensional

Objective

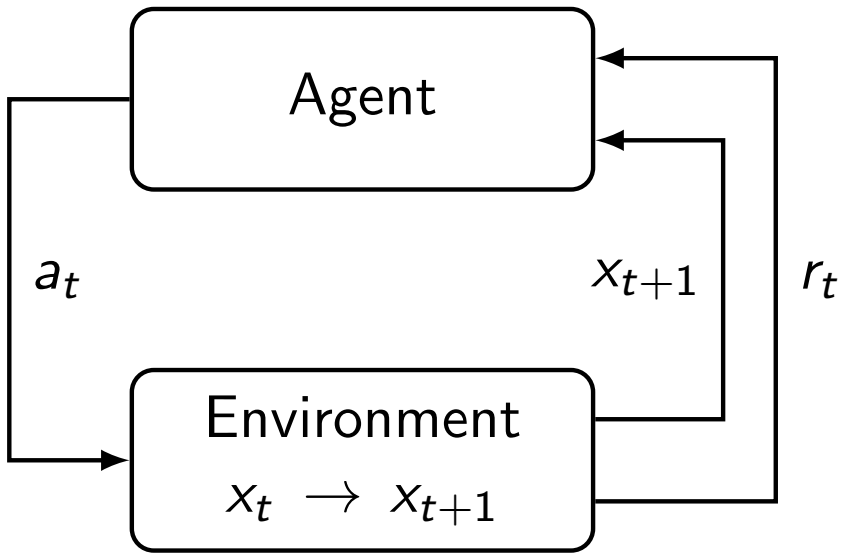
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Observations may not
provide full knowledge
of the underlying
state : $\omega_t \neq x_t$

Objective

From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Experience may be constrained
(e.g., not access to an accurate simulator or limited data)

Example of a Markov Decision Process (MDP)

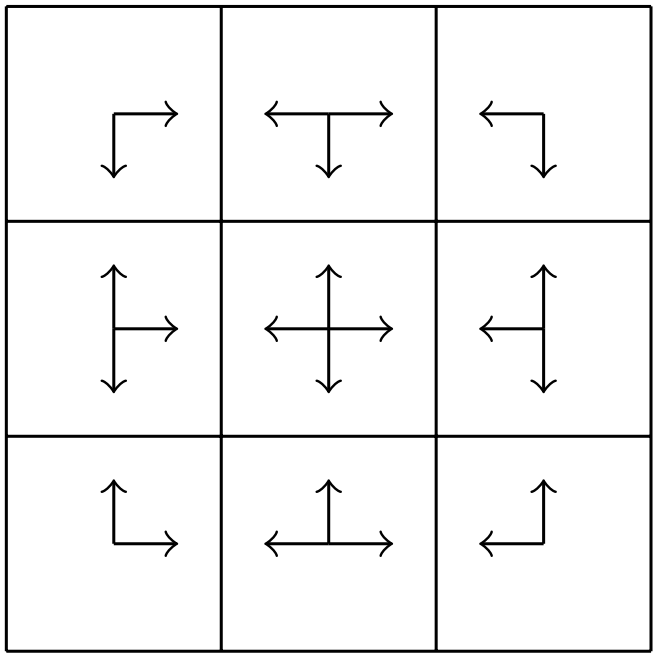
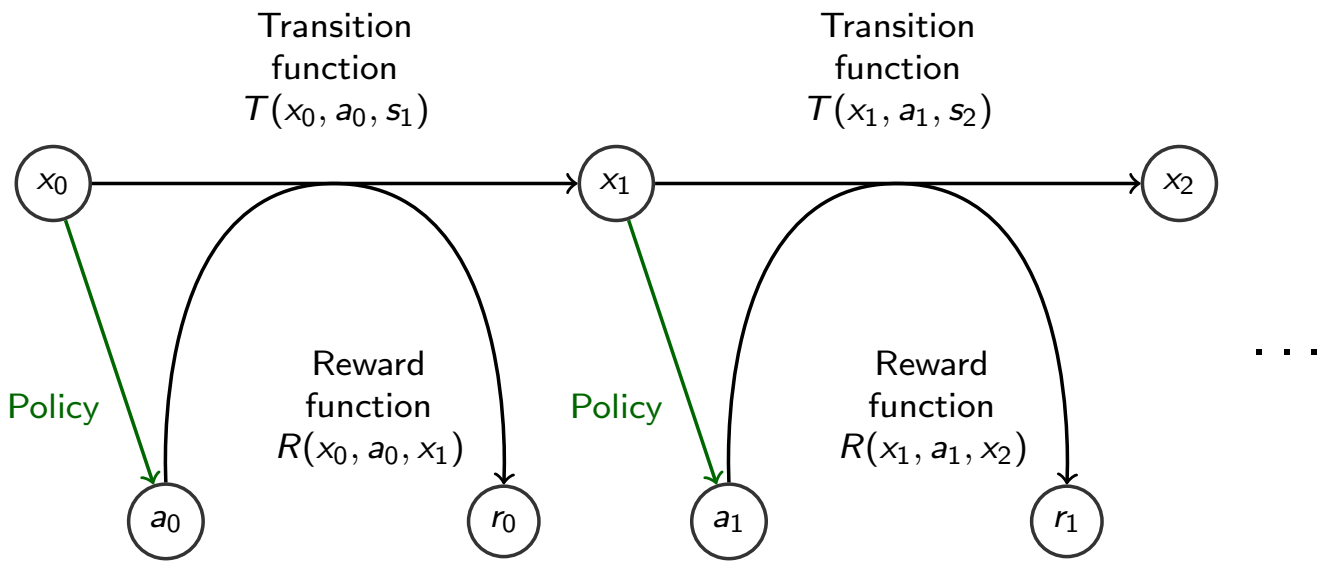


FIGURE – Representation of a (deterministic) mini grid-world with 9 discrete states and 4 discrete actions. The agent is able to move in the four directions, except when the agent is trying to get "out of the grid-world".

Definition of an MDP

An MDP can be defined as a 5-tuple $(\mathcal{X}, \mathcal{A}, T, R, \gamma)$ where :

- $\mathcal{X}$ is a finite set of states $\{1, \ldots, N_{\mathcal{X}}\}$,
- $\mathcal{A}$ is a finite set of actions $\{1, \ldots, N_{\mathcal{A}}\}$,
- $T : \mathcal{X} \times \mathcal{A} \to \mathbb{P}(\mathcal{X})$ is the transition function (set of conditional transition probabilities between states),
- $R : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathcal{R}$ is the reward function, where $\mathcal{R}$ is a continuous set of possible rewards in a range $R_{max} \in \mathbb{R}^+$ (e.g., $[0, R_{max}]$),
- $\gamma \in [0, 1)$ is the discount factor.



Performance evaluation

In an MDP $(\mathcal{X}, \mathcal{A}, T, R, \gamma)$, the discounted expected return $V^\pi(x) : \mathcal{X} \to \mathbb{R}$ ($\pi \in \Pi$, e.g., $\mathcal{X} \to \mathcal{A}$) is defined such that

$$V^\pi(x) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid x_t = x, \pi\right], \tag{1}$$

with $\gamma \in [0, 1)$.

From the definition of the (discounted) expected return, the optimal expected return can be defined as

$$V^*(x) = \max_{\pi \in \Pi} V^\pi(x). \tag{2}$$

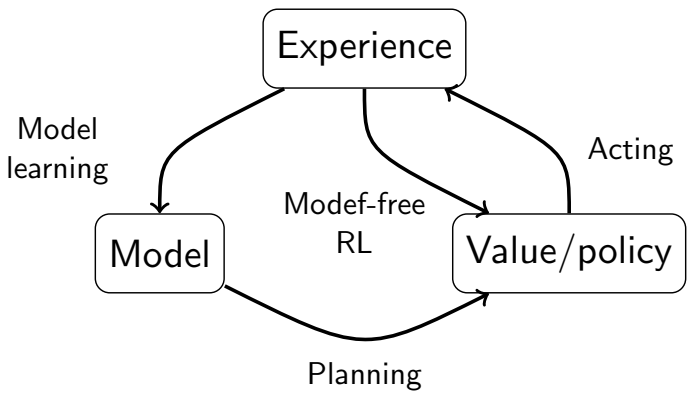
and the optimal policy can be defined as :

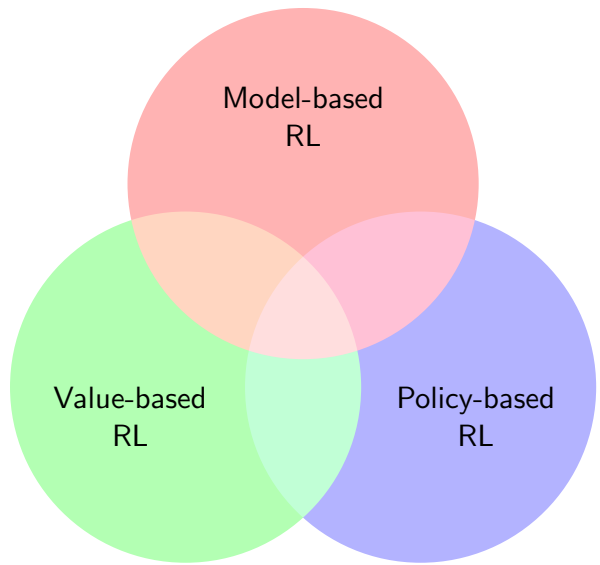
$$\pi^*(x) = \operatorname*{argmax}_{\pi \in \Pi} V^\pi(x). \tag{3}$$

Overview of the techniques used for finding the optimal policy π^*

In general, an RL agent may include one or more of the following components :

- a representation of a value function that provides a prediction of how good is each state or each couple state/action,
- a direct representation of the policy $\pi(x)$ or $\pi(x, a)$, or
- a model of the environment in conjunction with a planning algorithm.





Motivation for tree search techniques and model-based methods

Motivation for planning with tree search

Given that you're playing the crosses, what would be your next move ?

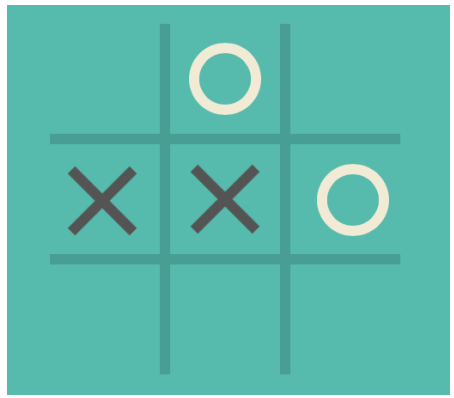


FIGURE – Illustration of a state in the tic-tac-toe game.

$\rightarrow$ how did you come up with that choice ?

Motivation

MCTS algorithms need (only) a generative model of the environment (i.e. model-based) :

$$x', r \sim G(x, a)$$

Advantages :

- it is possible to obtain samples without having the whole transition function for the model in an explicit form.
- it can learn a strong policy only where needed (from the current state x).
- it is useful for a sequence of decisions.

Motivation

- Tree search algorithms can be used along with different heuristics as well as model-free deep RL techniques.

$\rightarrow$ MCTS has been a key part of alpha Go for instance.

Monte-Carlo Tree Search methods

The overall idea is to estimate the action with the highest expected return.

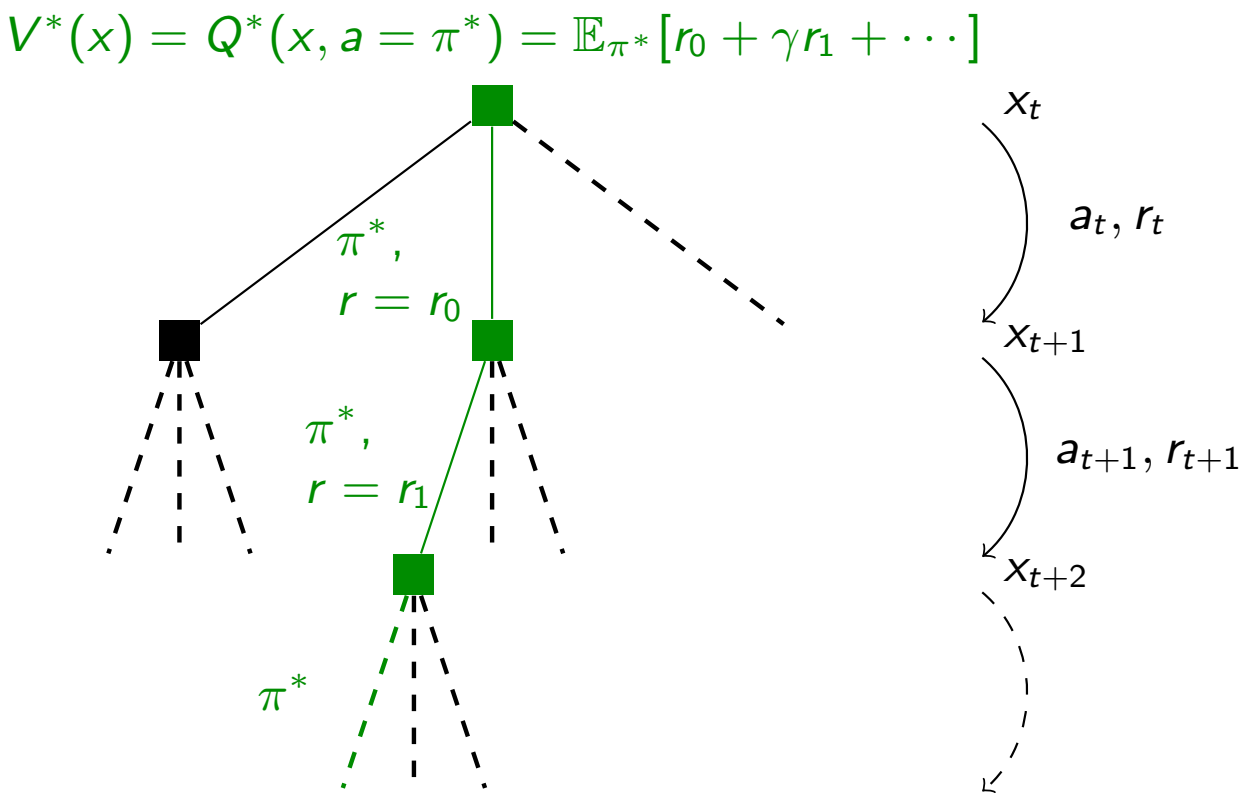


FIGURE – Illustration of model-based planning with tree search.

Monte-Carlo Tree Search methods

Monte-Carlo Tree Search (MCTS) is a key algorithm for finding a good policy trying out trajectories from a state x in an MDP.

It is a combination of

- Monte-Carlo methods, i.e. a class of computational algorithms that rely on repeated random sampling to obtain numerical results.
- Tree search, i.e. methods that allow exploring a tree data structure.

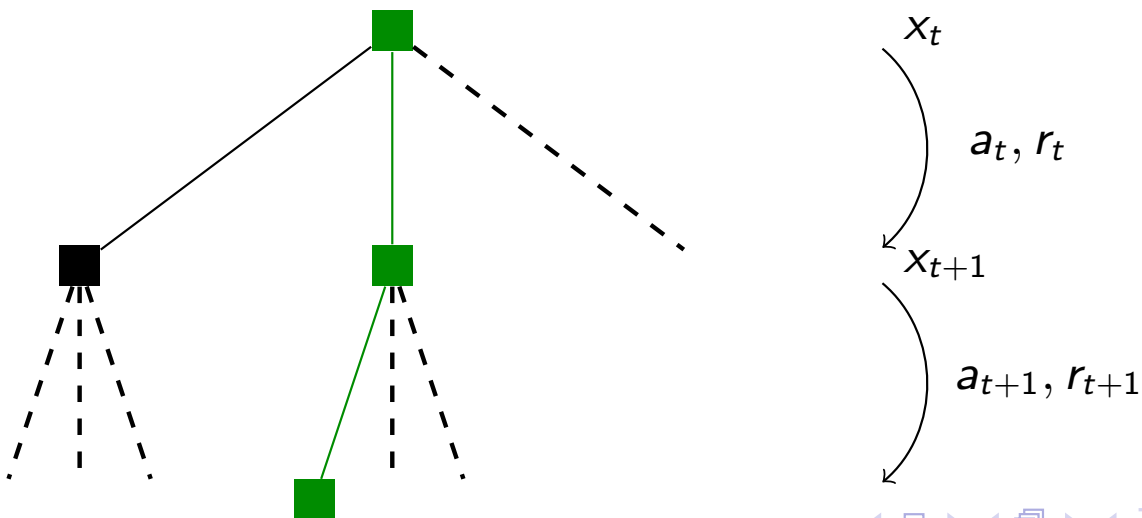
Description of the MCTS algorithm

Purely random Monte Carlo tree search

Purely random Monte Carlo tree search

1. Build a search tree with the current state of the agent at the root
2. Until convergence
   - Sample a trajectory with random actions (e.g. actions distributed uniformly)
   - Update estimates of the value functions where appropriate using already simulated episodes by backing up the values from the leafs of the tree to the root node :

$$Q(x,a) = \frac{\sum_{x'} n(x' \mid x,a)\left(r + \gamma \max_{a' \in \mathcal{A}} Q(x',a')\right)}{n(x,a)}. \qquad (4)$$



Monte Carlo tree search in n-players games

Monte Carlo tree search can be combined with the "*minimax*" / "*maximin*" approach. The *maximin* approach is a decision rule for maximizing the rewards for a worst case (maximum loss) scenario.

$\rightarrow$ well-suited for 2-players zero-sum games (or n-players) !

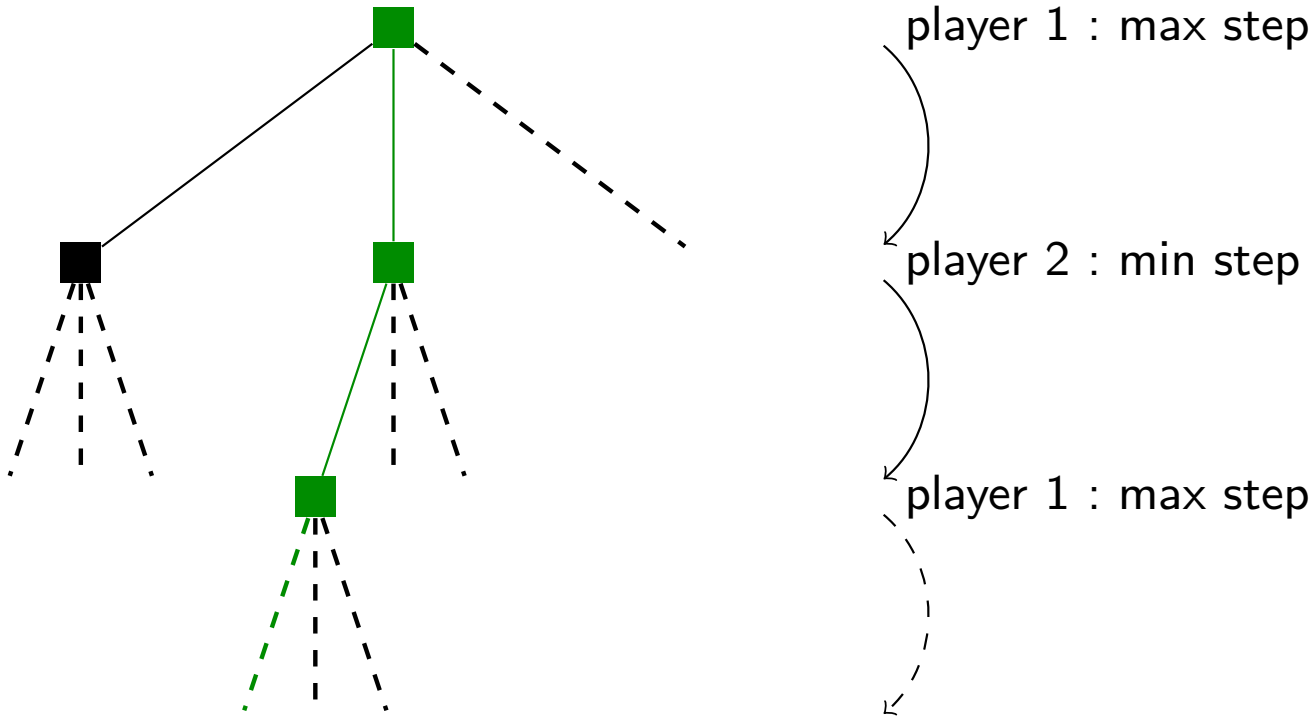


FIGURE – Illustration of tree search in 2-players zero-sum game.

Purely random Monte Carlo tree search

1. Build a search tree with the current state of the agent at the root
2. Until convergence
   - Sample a trajectory with random actions (e.g. actions distributed uniformly)
   - Update estimates of the value functions where appropriate using already simulated episodes by backing up the values from the leafs of the tree to the root node :

$$Q(x,a,\cdot) = \frac{\sum_{x'} n(x'|x,a) \left(r + \gamma \max_{a^{(p1)} \in \mathcal{A}_{p1}} \min_{a^{(p2)} \in \mathcal{A}_{p2}} Q(x',\cdot,\cdot) \right)}{n(x,a)}$$

Monte-Carlo Tree search

Good properties of algorithms seen thus far :

- The output will be correct within a certain probability (tradeoff between speed and accuracy).
- Pure Monte Carlo Game Search converges to the optimal policy (as the number of trajectories tends to infinity).

$\rightarrow$ simple MCTS algorithms results in strong play in several games.

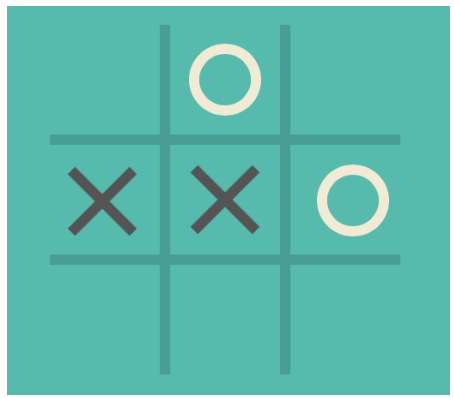


FIGURE – Tic tac toe can be solved with "simple" MCTS algorithms while having a reasonable computational budget.

Remaining challenge

Converges to the optimal policy

but

Remaining challenge

Converges to the optimal policy

but

it needs *a lot* of trial and errors when the complexity of the problem increases

Improvements on Purely random MCTS

FIGURE – Deep Blue (1997).

Is it possible to solve chess with the MCTS algorithm described above ?

Improvements on Purely random MCTS

FIGURE – Deep Blue (1997).

Is it possible to solve chess with the MCTS algorithm described above ?

Given an average depth d and an average number of possible moves w, an approximation for the size of the tree would be :
$w^d \approx 30^{85}$

MCTS

Improvements needed on pure Monte Carlo tree search if large state-action space :

- Sample more often the promising state-action pairs.
  - Find a good exploration/exploitation tradeoff to select the actions to expand
- For long time horizon : find a way to deal with long sequences of actions
  - cut the tree at some depth and estimate the value function at the leaf

NB : In the following, we consider environments where rewards are only on final states, e.g., win (+1) or lose (-1).

First option : heuristic evaluator

An heuristic evaluator is based on domain-specific knowledge and can guide the search :

- it can help in the exploration/exploitation (e.g. by reducing the breadth of search)
- it can provide values at the leaves of the search tree.

An evaluation function for chess might take the form :

$$c1 * \text{material} + c2 * \text{mobility} + c3 * \text{king safety} +$$

$$c4 * \text{center control} + c5 * \text{pawn structure...}$$

Second option : learning

Why learning ? In the game of Go for instance, simple heuristic evaluators are quite bad.

- We can reduce the breadth of search from data that we acquire online and we can estimate the value at the leaf with random rollouts (MCTS with UCT).
- We can make use of a learned estimate of the expected return to guide the search as well as estimate the values at the leafs (MCTS and function approximator such as deep learning).

Pause 10 min ?

More advanced exploration and exploitation techniques

MCTS with UCT

MCTS with UCT (upper confidence trees) provides a solution for

- exploration/exploitation by selecting the branches to expand in the selection step
- random rollout from the leafs to get an estimate of the win/lose.

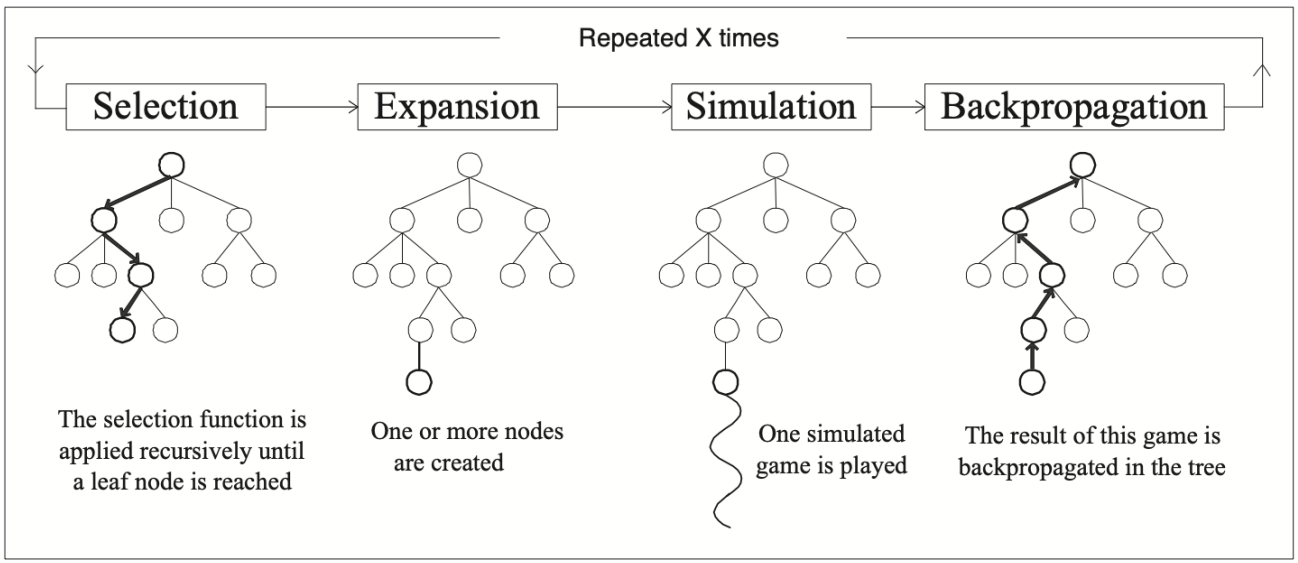


FIGURE – Illustration of MCTS (Chaslot et al. 2008).

Exploration and exploitation in UCT

In UCT, it is recommended to use the UCB rule and choose in each node of the game tree the move corresponding to the highest following expression

$$\underbrace{\frac{w_i}{n_i}}_{\text{exploitation}} + \underbrace{c\sqrt{\frac{\ln N_i}{n_i}}}_{\text{exploration}}.$$

Exploration is obtained with optimism in the face of uncertainty.

- w_i := number of wins for the node considered after the i-th move
- n_i := number of simulations for the child node (i.e. action)
- N_i := total number of simulations for the parent node
- c := exploration parameter, in practice usually chosen empirically (in a specific theoretical setting equal to $\sqrt{2}$)

MCTS with UCT

1. Selection : Used for nodes we have seen before : pick the action according to UCB
2. Expansion : Used when we reach a leaf node in the current tree : add one node
3. Simulation : Used from the new leaf node : perform a random rollout (without bothering with UCB)
4. Backtrack : After reaching a terminal node : update value and visits for states expanded in selection and expansion

MCTS with UCT

MCTS with UCT provides a solution for

- exploration/exploitation by selecting the branches to expand in the selection step
- random rollout from the leafs to get an estimate of the win/lose.

MCTS with UCT

MCTS with UCT provides a solution for

- exploration/exploitation by selecting the branches to expand in the selection step
- random rollout from the leafs to get an estimate of the win/lose.

However,

- If the actions space is large, the breath of search can still be huge (computationally expensive)
- At the leaf nodes of the tree, purely random rollouts are not always ideal (possible bias+variance).

Alpha Go

FIGURE – Alpha Go (2016).

Beyond UCT : learning with generalization

- → To reduce depth and breath, we could make use of a learned heuristic of $Q(x, a)$. We thus need generalization between similar states in the search tree.
- → If we use rollouts, we could make use of a smarter heuristic than random : $\rightarrow$ rollout policy $\pi_r(x, a)$

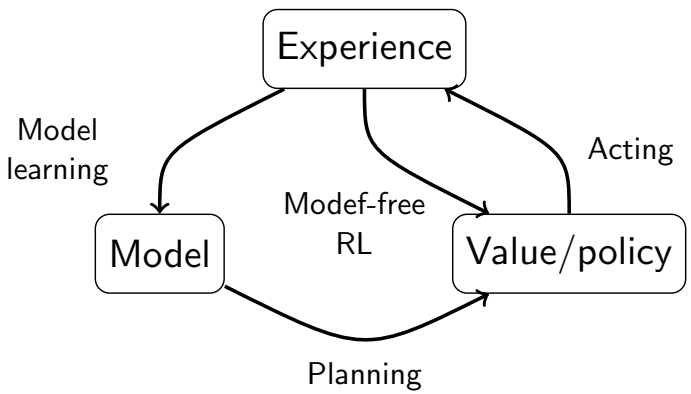
The curse of dimensionality is treated thanks to the generalization achieved by the deep learning function approximators.

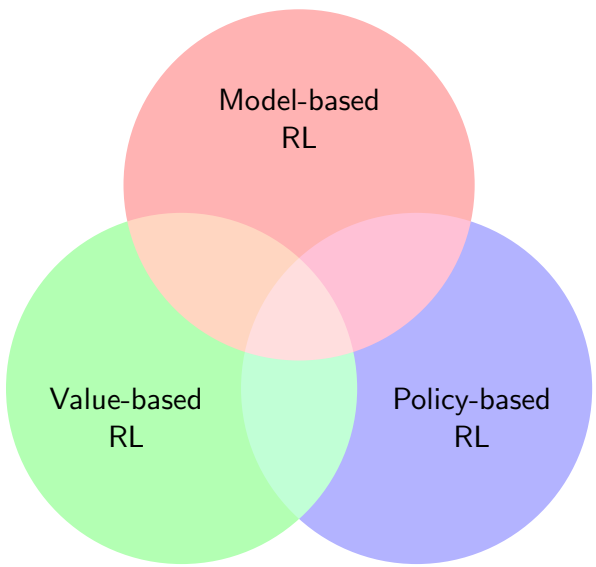
Combining MCTS and deep learning

Reinforcement learning + function approximators

In general, an RL agent may include one or more of the following components :

- a representation of a value function that provides a prediction of how good is each state or each couple state/action,
- a direct representation of the policy $\pi(x)$ or $\pi(x, a)$, or
- a model of the environment in conjunction with a planning algorithm.





Deep learning has brought its generalization capabilities to RL.

Combining MCTS and deep learning : a sneak peak into the alpha go case

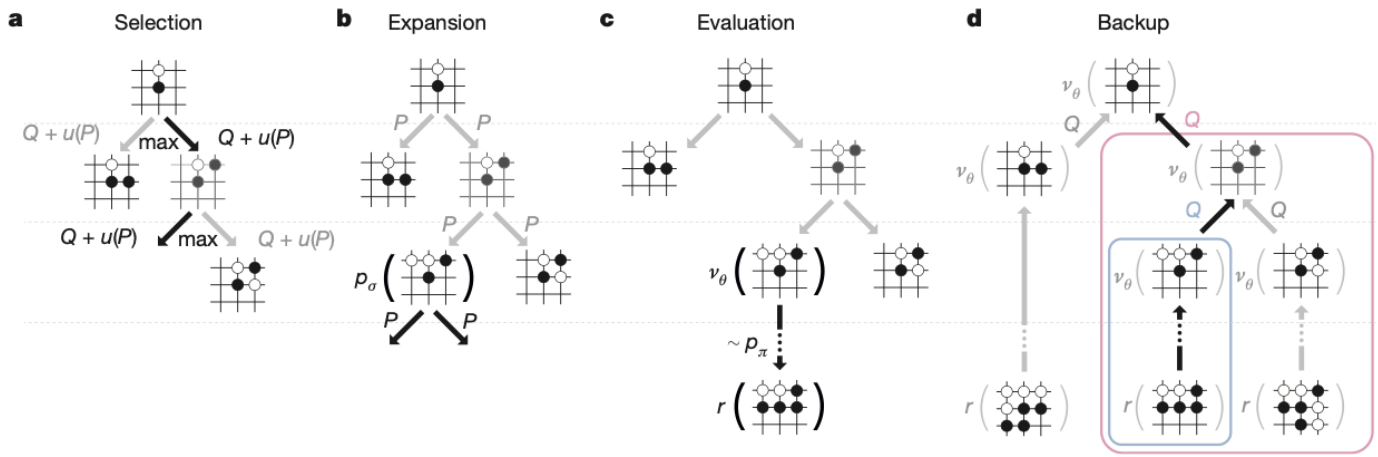


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

FIGURE – Illustration of alpha Go MCTS (Silver, et al. 2016)

The alpha go case

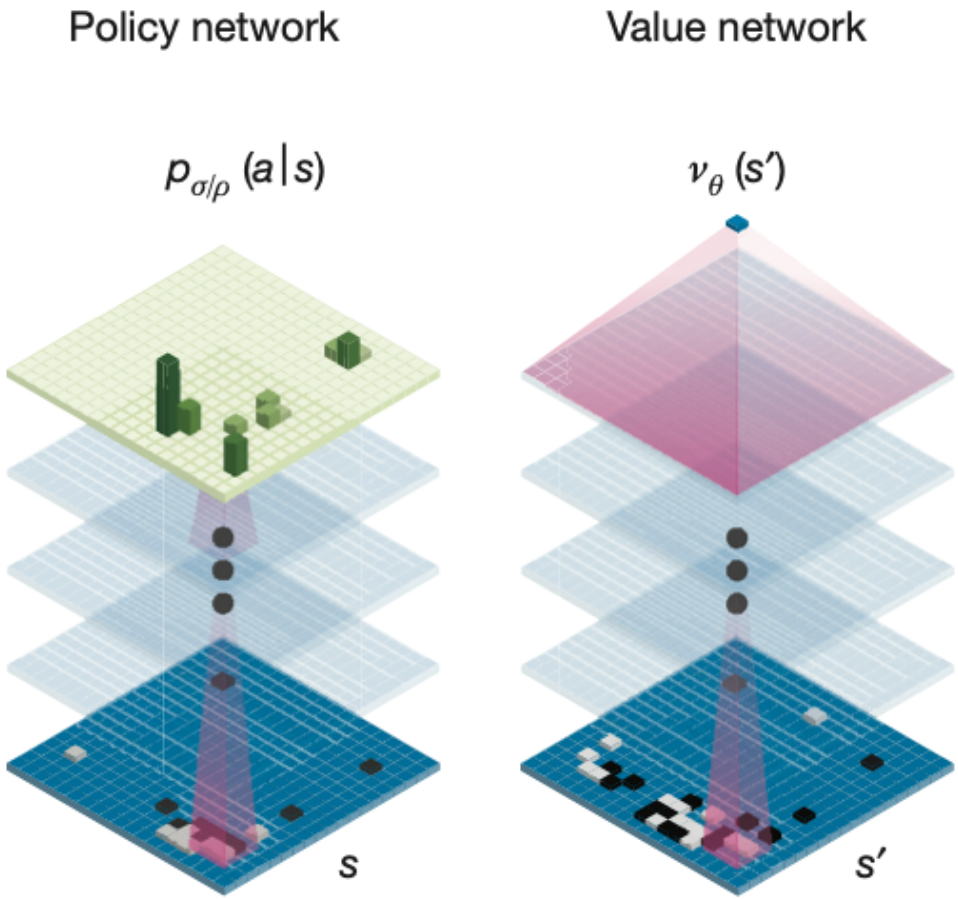


FIGURE – Illustration of alpha Go policy network and Value network (Silver, et al. 2016)

The alpha go case

FIGURE – Illustration of alpha Go MCTS (Silver, et al. 2016)

Discussion on model-based methods

Discussion on model-based methods

The respective strengths of the model-free versus model-based approaches depend on different factors.

- If the agent does not have access to a generative model of the environment, the learned model will have some inaccuracies.
- Second, a model-based approach requires working in conjunction with a planning algorithm, which is often computationally demanding.
- Third, for some tasks, the model of the environment may be learned more efficiently due to the particular structure of the task.

Summary of the methods

- Purely random MCTS.
  - Develop the tree with purely random trajectories
- MCTS with UCT.
  - Develop the tree with an exploration-exploitation tradeoff given by the UCT formula
  - Estimate the values at the leafs with random rollouts
- MCTS guided with deep learning.
  - Guide the exploration/exploitation tradeoff with policy network (deep learning)
  - Estimate the values at the leaf with value network (deep learning), possibly in combination with rollouts

Conclusion

Conclusion

- Why planning is interesting
- Basic MCTS algorithm
- The need for a good exploration-exploitation tradeoff : e.g. MCTS algorithm with UCT
- The need for generalization (both for selecting actions and for estimating the values at the leaf) : sneak peak into the alpha-go algorithm

Next Fridays, we'll delve into a full introduction to deep RL, and model-free techniques (value-based and policy based).

Further reading (optional)

- Coulom, Rémi. "Efficient selectivity and backup operators in Monte-Carlo tree search." In International conference on computers and games, pp. 72-83. Springer, Berlin, Heidelberg, 2006.
- Kocsis, Levente, and Csaba Szepesvari. "Bandit based monte-carlo planning." European conference on machine learning. Springer, Berlin, Heidelberg, 2006.
- Browne, Cameron B., et al. "A survey of monte carlo tree search methods." IEEE Transactions on Computational Intelligence and AI in games 4.1 (2012) : 1-43.
- Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016) : 484-489.

Questions ?