Sequential Decision Making

Lecture 9 : Monte Carlo Tree Search

Emilie Kaufmann

M2 Data Science, 2022/2023

Outline

Monte-Carlo Tree Search

MCTS is a family of methods that adaptively explore the tree of possible next states in a given state s_1, in order to find the best action in s_1.

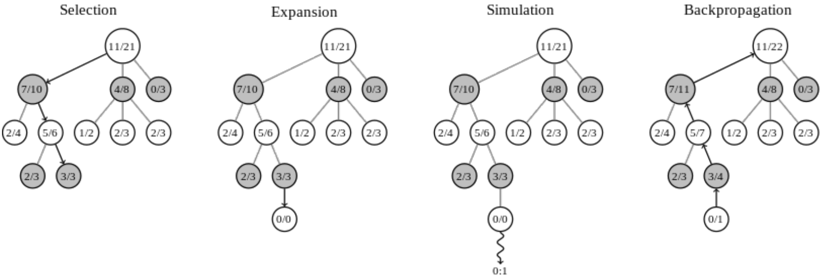


FIGURE – A generic MCTS algorithm for a game

Outline

The UCT algorithm

Bandit-Based Monte-Carlo planning : to select a path in the tree, run a bandit algorithm each time a children (next action) needs to be selected

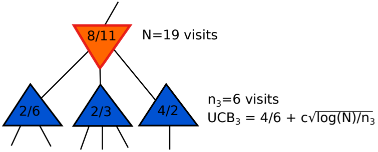
UCT = UCB for Trees [Kocsis and Szepesvári, 2006]

UCT in a Game Tree

In a MAX node s (= root player move), select an action

$$\underset{a \in \mathcal{C}(s)}{\operatorname{argmax}} \ \frac{S(s,a)}{N(s,a)} + c\sqrt{\frac{\ln\left(\sum_b N(s,b)\right)}{N(s,a)}}$$

$N(s,a)$: number of visits of (s,a)
$S(s,a)$: number of visits of (s,a) ending with the root player winning



The UCT algorithm

Bandit-Based Monte-Carlo planning : to select a path in the tree, run a bandit algorithm each time a children (next action) needs to be selected

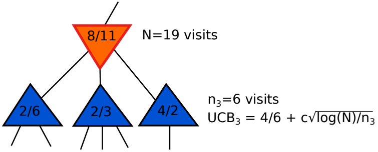
UCT = UCB for Trees [Kocsis and Szepesvári, 2006]

UCT in a Game Tree

In a MIN node s (= adversary move), select an action

$$\underset{a \in \mathcal{C}(s)}{\operatorname{argmin}} \ \frac{S(s,a)}{N(s,a)} - c\sqrt{\frac{\ln\left(\sum_b N(s,b)\right)}{N(s,a)}}$$

$N(s,a)$: number of visits of (s,a)
$S(s,a)$: number of visits of (s,a) ending with the root player winning



The UCT algorithm

Bandit-Based Monte-Carlo planning : to select a path in the tree, run a bandit algorithm each time a children (next action) needs to be selected

UCT = UCB for Trees [Kocsis and Szepesvári, 2006]

UCT in a Game Tree

In a MAX node s (= root player move), select an action

$$\underset{a\in\mathcal{C}(s)}{\text{argmax}} \ \frac{S(s,a)}{N(s,a)} + c\sqrt{\frac{\ln\left(\sum_b N(s,b)\right)}{N(s,a)}}$$

$N(s,a)$: number of visits of (s,a)
$S(s,a)$: number of visits of (s,a) ending with the root player winning

When a leaf (or some maximal depth) is reached :

- a playout is performed (play the game until the end with a simple heuristic, or produce a random evaluation of the leaf position)
- the outcome of the playout (typically $1/0$) is stored in all the nodes visited in the previous trajectory

The UCT algorithm

- first good AIs for Go where based on variants on UCT
- it remains a heuristic (no sample complexity guarantees, parameter c fined-tuned for each application)
- many variants have been proposed

[Browne et al., 2012]

Outline

Alpha Zero

AlphaZero learns a good policy by using a MCTS algorithm guided by a neural network
$\neq$ pure play-out based MCTS

Input

A neural network predicting a policy $\boldsymbol{p} \in \Delta(\mathcal{A})$ and a value $v \in \mathbb{R}$ from the current state s : $(\boldsymbol{p}, v) = f_\theta(s)$.

The MCTS algorithm maintains for each visited state/action the counts and cumulated values + a vector of prior action probabilities :

$$\{N(s,a), S(s,a), P(s,a)\}$$

Alpha Zero

AlphaZero learns a good policy by using a MCTS algorithm guided by a neural network

$\neq$ pure play-out based MCTS

Input

A neural network predicting a policy $\boldsymbol{p} \in \Delta(\mathcal{A})$ and a value $v \in \mathbb{R}$ from the current state s : $(\boldsymbol{p}, v) = f_\theta(s)$.

The MCTS algorithm maintains for each visited state/action the counts and cumulated values + a vector of prior action probabilities :

$$\{N(s,a), S(s,a), P(s,a)\}$$

Selection step : in some state s, choose the next action to be

$$\underset{a \in \mathcal{C}(s)}{\mathrm{argmax}} \left[\frac{S(s,a)}{N(s,a)} + c \times P(s,a) \frac{\sqrt{N(s)}}{1 + N(s,a)} \right]$$

for some (fine-tuned) constant c.

Alpha Zero

AlphaZero learns a good policy by using a MCTS algorithm guided by a neural network
$\neq$ pure play-out based MCTS

Input

A neural network predicting a policy $\boldsymbol{p} \in \Delta(\mathcal{A})$ and a value $v \in \mathbb{R}$ from the current state s : $(\boldsymbol{p}, v) = f_\theta(s)$.

The MCTS algorithm maintains for each visited state/action the counts and cumulated values + a vector of prior action probabilities :

$$\{N(s,a), S(s,a), P(s,a)\}$$

Expansion step : once a leaf s_L is reached, compute $(\boldsymbol{p}, v) = f_\theta(s_L)$.

- Set v to be the value of the leaf
- For all possible next actions b :
  - ➜ initialize the count $N(s_L, b) = 0$
  - ➜ initialize the prior probability $P(s_L, b) = \boldsymbol{p}_b$ (possibly add some noise)

Alpha Zero

AlphaZero learns a good policy by using a MCTS algorithm guided by a neural network
$\neq$ pure play-out based MCTS

Input

A neural network predicting a policy $\boldsymbol{p} \in \Delta(\mathcal{A})$ and a value $v \in \mathbb{R}$ from the current state s : $(\boldsymbol{p}, v) = f_\theta(s)$.

The MCTS algorithm maintains for each visited state/action the counts and cumulated values + a vector of prior action probabilities :

$$\{N(s,a), S(s,a), P(s,a)\}$$

Back-up step : for all ancestor s_t, a_t in the trajectory that end in leaf s_L,

$$\begin{aligned} N(s_t, a_t) &\leftarrow N(s_t, a_t) + 1 \\ S(s_t, a_t) &\leftarrow S(s_t, a_t) + v \end{aligned}$$

Alpha Zero

AlphaZero learns a good policy by using a MCTS algorithm guided by a neural network
$\neq$ pure play-out based MCTS

Input

A neural network predicting a policy $\boldsymbol{p} \in \Delta(\mathcal{A})$ and a value $v \in \mathbb{R}$ from the current state s : $(\boldsymbol{p}, v) = f_\theta(s)$.

The MCTS algorithm maintains for each visited state/action the counts and cumulated values + a vector of prior action probabilities :

$$\{N(s,a), S(s,a), P(s,a)\}$$

Output of the planning algorithm ? select an action a at random according to

$$\pi(a) = \frac{N(s_0,a)^{1/\tau}}{\sum_b N(s_0,b)^{1/\tau}}$$

for some (fine-tuned) temperature τ.

Training the neural network

- In AlphaGo, f_θ was trained on a database of games played by human
- In AlphaZero, the network is trained using only self-play

[Silver et al., 2016, Silver et al., 2017]

Let θ be the current parameter of the network $(\boldsymbol{p}, v) = f_\theta(s_L)$.

1. generate N games where each player uses MCTS(θ) to select the next action a_t (and output a probability over actions π_t)

$$\mathcal{D} = \bigcup_{i=1}^{\text{Nb games}} \left\{ \left(s_t, \pi_t, \pm r_{T_i} \right) \right\}_{i=1}^{T_i}$$

   T_i : length of game i, $r_{T_i} \in \{-1, 0, 1\}$ outcome of game i for one player

2. Based on a sub-sample of $\mathcal{D}$, train the neural network using stochastic gradient descent on the loss function

$$L(s, \boldsymbol{\pi}, z; \boldsymbol{p}, v) = (z - v)^2 - \boldsymbol{\pi}^\top \ln(\boldsymbol{p}) + c\|\theta\|^2$$

A nice actor-critic architecture

AlphaZero alternates between

- **The actor** : MCTS(θ)
  generates trajectories guided by the network f_θ but still exploring

➜ act as a policy improvement
($N = 25000$ games played, each call to MCTS uses 1600 simulations)

- **The critic** : neural network f_θ
  updates θ based on trajectories followed by the critic

➜ evaluate the actor's policy

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012).
A survey of monte carlo tree search methods.
IEEE Transactions on Computational Intelligence and AI in games,, 4(1) :1–49.

Kocsis, L. and Szepesvári, C. (2006).
Bandit based monte-carlo planning.
In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg. Springer-Verlag.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016).
Mastering the game of go with deep neural networks and tree search.
Nature, 529 :484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017).
Mastering the game of go without human knowledge.
Nature, 550 :354–.