An Introduction to Monte-Carlo Tree Search

Antoine Cornuéjols

AgroParisTech – INRAe Paris-Saclay

antoine.cornuejols@agroparistech.fr

Plan

1. Limites de l'approche classique
2. Évaluation par Monte-Carlo
3. Le compromis Exploration vs. Exploitation : algorithmes de bandits
4. Approche ε-greedy
5. UCT = MCTS + UCB
6. Illustrations
7. AlaphaGo Zero

Conventional game tree search

1. **Perfect-information** games

   All aspects of the states are **fully observable**

2. Technique: **MinMax** algorithm (with Alpha-Beta pruning)

   Effective for

   - **Modest branching** factor
   - When a **good evaluation function** is available

Problems with Go

- The **branching factor is large**

  $b \approx 250$ on average ; depth > 200 moves

- We do **not** know a **good evaluation function**

  Similar looking positions may have completely different outcomes

→ Alpha-Beta gives **weak to intermediate level** of play

Plan

1. Limites de l'approche classique
2. Évaluation par Monte-Carlo
3. Le compromis Exploration vs. Exploitation : algorithmes de bandits
4. Approche ε-greedy
5. UCT = MCTS + UCB
6. Illustrations
7. AlaphaGo Zero

Naïve approach: but interesting

Basic Monte-Carlo simulations

1. Simulate games using **random** moves
2. **Score** each game at the end
3. Store winning **statistics**
4. **Play** move with best winning percentage
5. Repeat

Use this a the **evaluation function**,

hoping it will ***preserve difference*** *between a good position and a bad one*

Limits

- **Use simulations directly as an evaluation function** for $\alpha\beta$
- Problems
  - Single simulation gives **very noisy evaluation**: 0/1 signal
  - **Running many simulations is required** for each position (move to consider)
  - Illustration
    - Typical speed of **chess** programs = 1 million eval/sec ; 30 moves to consider
      => 33,000 eval/move
    - **Go**: 1 million eval/sec ; 250 moves to consider ;
      => 4,000 eval/move.sec (not a lot for a complicated game)

→ Monte-Carlo was ignored for over 10 years in Go

Limits

- **Does not allocate search and evaluation wisely**: based on promise of the positions
- Evaluation is costly
  - We would like to have **precise evaluation in the promising regions** of the search space. It does not matter to be precise elsewhere as long as we are certain it is worthless.

→ The exploration vs. exploitation tradeoff

Plan

1. Limites de l'approche classique
2. Évaluation par Monte-Carlo
3. Le compromis Exploration vs. Exploitation : algorithmes de bandits
4. Approche ε-greedy
5. UCT = MCTS + UCB
6. Illustrations
7. AlaphaGo Zero

Idea

Use results of previous simulations to guide growth of the game tree

Politique de « sélection »

- **Exploitation**: play the **seemingly best move**
- **Exploration**: explore moves where the **uncertainty** is **highest**

→ Theory of bandits

Multi-Armed Bandit problem: illustration

Figure

Multi-Armed Bandit problem

Assumptions

- Choice among several arms
- Each arm pull is **independent** of other pulls
- Each arm pull is determined by a **distribution of unknown mean**

Which arm has the best average payoff?

Multi-Armed Bandit problem: illustration

A

P(A wins) = 60%

B

P(B wins) = 55%

C

P(C wins) = 40%

Consider a row with *three slot machines*

- Each pull of an arm is either
  - A **win**: payoff **1**
  - A **loss**: payoff **0**
- Here A is the best arm → but we do not know that

Plan

1. Limites de l'approche classique
2. Évaluation par Monte-Carlo
3. Le compromis Exploration vs. Exploitation : algorithmes de bandits
4. Approche ε-greedy
5. UCT = MCTS + UCB
6. Illustrations
7. AlaphaGo Zero

Multi-Armed Bandit problem: illustration

A

P(A wins) = 60%

B

P(B wins) = 55%

C

P(C wins) = 40%

We want to **minimize** the "**regret**" = loss wrt. the best sequence of pulls (if we had known the best arm at the start)

→ Need to **balance** exploration and exploitation

↓ ***Uniform*** *policy*

↓ ***Greedy*** *policy*

The ε-greedy algorithm

- **By default**, actions are chosen greedily.

  The action with the highest estimated reward is the selected action.

- However, at each time step, an action **may instead be selected at random** from the set of all possible actions.

  A random action is chosen with a probability 'ε' (Epsilon).

The ε-greedy algorithm

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Q(a) : estimate of the value of arm a

N(a) : number of draws of arm a

R : reward

From Sutton & Barto (2018). « **Reinforcement learning. An introduction** » (2nd edition)

The ε-greedy algorithm

Illustration

- Suppose 5 arms with mean rewards :
  - Arm 1 : **6** Arm 2 : **4** Arm : 3 : **8** Arm 4 : **12** Arm 5 : **10**

The optimal value for ε seems to be ~ **0.2**

The ε-greedy algorithm

Figure 2.1: An example bandit problem from the 10-armed testbed. The true value $q_*(a)$ of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean $q_*(a)$, unit-variance normal distribution, as suggested by these gray distributions.

From Sutton & Barto (2018). « **Reinforcement learning. An introduction** » (2nd edition)

The ε-greedy algorithm : choice for ε

Figure 2.2: Average performance of ε-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

...

" Regret "

- **Optimal** action $$a_* = \arg\max_a \mathbb{E}[R_t | A_t = a]$$

- **Regret** *L* after *T* time steps

$$L = \underbrace{T.\mathbb{E}[R_t | A_t = a_*]}_{\text{If I had known the } \textbf{best arm}} - \underbrace{\sum_{t=1}^{T} \mathbb{E}[R_t | A_t = a]}_{\text{The } \textbf{choices} \text{ I actually made}}$$

$$L = \underbrace{T.\mathbb{E}[R_t|A_t = a_*]}_{\text{If I had known the best arm}} - \underbrace{\sum_{t=1}^{T} \mathbb{E}[R_t|A_t = a]}_{\text{The choices I actually made}}$$

$$\boxed{\varepsilon T \frac{(k-1)}{k} \Delta \;\leq\; L \;\leq\; \varepsilon T \frac{(k-1)}{k} \Delta'}$$

Linear in time *T*

k arms

Δ = difference between **best** arm and the **next** one

Δ' = difference between **best** arm and the **worst** one

“ Regret ”

- **Regret** *L* after *T* time steps

$$\varepsilon T \frac{(k-1)}{k} \Delta \;\leq\; L \;\leq\; \varepsilon T \frac{(k-1)}{k} \Delta'$$

$$\boxed{\varepsilon T \frac{(k-1)}{k} \Delta \;\leq\; L \;\leq\; \varepsilon T \frac{(k-1)}{k} \Delta'}$$

Linear in time *T*

k arms

Δ = difference between **best** arm and the **next** one

Δ' = difference between **best** arm and the **worst** one

Good for a naïve approach. But we can do better

See below the **UCB algorithm**

Plan

1. Limites de l'approche classique
2. Évaluation par Monte-Carlo
3. Le compromis Exploration vs. Exploitation : algorithmes de bandits
4. Approche ε-greedy
5. UCT = MCTS + UCB
6. Illustrations
7. AlaphaGo Zero

The **Upper Confidence Bound** algorithm

Principle: Optimism in the face of uncertainty

– **Choose the action** as if the environment is as nice as is **plausibly** possible

The unknown mean payoff of each arm is as large as plausibly possible based on the data that has been observed

Intuition. One of two things can happen

1. The **optimism was justified**: the learner is acting optimally
2. The optimism was **not justified**: the agent takes some action that he believes might give large reward when in fact it does not.

   If this happens sufficiently often, then **the learner will learn what is the true payoff of this action and not choose it in the future**

What "plausible" means

If $X_1, X_2, \ldots, X_n$ are independent and 1-subgaussian (which means that $\mathbb{E}(X_i) = 0$), and $\hat{\mu} = \sum_{i=1}^{n} X_t/n$, then:

$$\mathbf{P}(\hat{\mu} \geq \varepsilon) \;\leq\; \exp(-n\varepsilon^2/2)$$

Equating the right-hand side with δ and solving for ε leads to

$$\mathbf{P}\left\{\hat{\mu} \;\geq \sqrt{\frac{2}{n}\log\left(\frac{1}{\delta}\right)}\right\} \;\leq\; \delta$$

Suggests a definition of "as large as plausibly possible"

Choice of a good action

When the agent is deciding what to do in round t, it has observed $T_i(t-1)$ samples from arm i and observed rewards with an empirical mean of $\hat{\mu}(t-1)$ for it. Then **the agent should choose the action** i that maximizes:

$$i \;=\; \underset{j \,\in\, \text{actions}}{\text{ArgMax}} \left\{ \hat{\mu}_j(t-1) \;+\; \sqrt{\frac{2}{T_j(t-1)} \log\left(\frac{1}{\delta}\right)} \right\}$$

We should explore more the arms with highest mean
or not well explored

δ is called the "confidence level" and different choices lead to different algorithms. We will take: $1/\delta \;=\; f(t) \;=\; 1 \;+\; t \log^2(t)$

...

Upper Confidence Bound (UCB)

UCB1 formula [Auer et al. 2002]

- The "true value" of arm *i* is expected to be in some confidence interval around v_i

$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

value estimate

tunable parameter

total number of trials

num trials for arm i

Upper Confidence Bound (UCB)

UCB1 formula [Auer et al. 2002]

- Policy
  1. First, try each arm once
  2. Then **at each time step**:

     Choose arm *i* that maximizes the UCB1 formula

$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

value estimate

tunable parameter

total number of trials so far

num trials for arm i

- Let us look at a very simple scenario with **two arms**
  - **Arm 1** with mean value of **6**
  - **Arm 2** with mean value of **4**

Upper Confidence Bound (UCB)

- The **solid part** of each bar represents the **exploration** part of the equation, therefore diminishing with increasing number of tests
- The **shaded part** of the bar represents the estimate of each arm's actual value, which converges to the true value

Upper Confidence Bound (UCB)

Here, the problem is simple and almost no exploration is needed

Upper Confidence Bound (UCB)

Looking more carefully, a small value of exploration (e.g. $C \sim 0.7$) yields the best result (here). The theory would favor $C = \sqrt{2}$

Upper Confidence Bound (UCB)

- Au début, les 5 bras sont explorés, puis ensuite c'est pratiquement toujours le bras 4 qui est tiré

Upper Confidence Bound (UCB)

- Presque tout le regret vient de l'exploration des 5 bras au début, puis il varie approximativement en log(n)

Theoretical properties of UCB1

The main question: rate of convergence to optimal arm

- Typical goal: achieve a regret of $\mathcal{O}(\log n)$ for n trials
- For many kinds of problems, it is **not possible to do better**
- **UCB1 is a simple algorithm** that achieves this asymptotic bound for many input distributions
- Huge amount of literature on different bandit problems and their properties

UCB vs. ε-greedy

…

Monte Carlo Tree Search

- Au lieu de se concentrer sur **un seul « bandit »**

- MCTS réalise **l'exploration d'arbres MinMax**
  - Chaque **successeur** est considéré comme un bandit
  - Une **séquence de bandits** est explorée à chaque épisode

- Contrairement à l'algorithme MinMax
  - MCTS réalise une exploration à **profondeur** et **largeur** variables

MCTS

- A generic way of exploring a game tree
- Four steps
  1. Selection
     - Of a leaf to expand
  2. Expansion
  3. Simulation (roll out)
  4. Back-propagation
     - Updating the value of each ancestor node of the expanded leaf

Upper Confidence Tree = MCTS + UCB

From Sutton & Barto (2018). « **Reinforcement learning. An introduction** » (2nd edition)

Upper Confidence Tree = MCTS + UCB

- How to perform the **selection**?

- Answer: use **UCB**

- Upper Confidence Tree:

  use a look-ahead tree with **selection guided by** UCB and exploration/**evaluation** performed by Monte-Carlo sampling

Upper Confidence Tree = MCTS + UCB

1. Tree traversal
2. Node expansion
3. Rollout (random simulations)
4. Backpropagation

Plan

1. Limites de l'approche classique
2. Évaluation par Monte-Carlo
3. Le compromis Exploration vs. Exploitation : algorithmes de bandits
4. Approche e-greedy
5. UCT = MCTS + UCB
6. Illustrations
7. AlaphaGo Zero

Upper Confidence Tree = MCTS + UCB

1. **Selection** NOT using UCB (here)

Max

Max **looses** 37 out of 100 iterations

Min

Max **wins** 60 out of 79 iterations, ...

Max

Max **looses** 16 out of 53 iterations,
hence wins 37 times out of 53

Min

Max **wins** 27 out of 35 iterations, ...

From [Russel & Norvig (2021). *Artificial Intelligence. A modern approach*. 4th Ed.]

Upper Confidence Tree = MCTS + UCB

2. Node **expansion** / rollout

(b) Expansion and simulation

Min = “black”

From [Russel & Norvig (2021). *Artificial Intelligence. A modern approach*. 4th Ed.]

Upper Confidence Tree = MCTS + UCB

4. **Backpropagation**

From [Russel & Norvig (2021). *Artificial Intelligence. A modern approach*. 4th Ed.]

Upper Confidence Tree = MCTS + UCB

- Note that the **node selection could have been different** according to the value of *C* in:

$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

value estimate

tunable parameter

total number of trials

num trials for arm i

UCT : the **algorithm**

Tree traversal and node expansion

Procedure Rollout

Rollout(s_i)

- **Loop until** s_i is a **terminal state**
  1. a_i = random(available_actions(s_i))
  2. s_i := simulate(a_i, s_i)
- Return value(s_i)

Here, an example where the return is not win or loose, but a number

Worked out example:

Example

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i

s_0

$$t_0 = 0$$
$$n_0 = 0$$

Worked out example: 1st iteration

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i

Choice of action

Worked out example: 1st iteration

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i

$UCB1(s_1) = \infty$ $\quad$ $UCB1(s_2) = \infty$

Choice of an action

Choice of action a_1

Worked out example: 1st iteration

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i

Choice of action

$UCB1(s_1) = \infty$ $\quad$ $UCB1(s_2) = \infty$

Choice of action a_1 not visited yet

Rollout

Worked out example: 1st iteration

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i

Choice of action

Rollout

Worked out example: 1st iteration

Example

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i

Worked out example: 2nd iteration

Example

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i

Choice of an action

$$UCB1(s_1) = 20 + 2\sqrt{\frac{\ln 1}{1}} = 20 \qquad UCB1(s_2) = \infty$$

Worked out example: 2nd iteration

Example

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i

Worked out example: 2nd iteration

Example

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i

Back-propagation

Worked out example: 3rd iteration

Example

Node expansion

Worked out example: 3rd iteration

Example

Node expansion

$$UCB1(s_3) = \infty \qquad UCB1(s_4) = \infty$$

Worked out example: 3rd iteration

Worked out example: 3rd iteration

Worked out example: 4th iteration

12.10

Worked out example: 4th iteration

Worked out example: 4th iteration

Worked out example: 4th iteration

Conclusions sur UCT

On continue le processus jusqu'à épuisement des ressources calcul allouées

+ Preuve de **convergence** vers MinMax, mais lent dans la version de base montrée ici
+ Pas besoin de **fonction d'évaluation**
+ Très bon quand le **facteur de branchement** est **important**. Contrôle bien le compromis exploration vs. Exploitation
+ Algorithme « **anytime** »

Conclusions (2)

- UCT (= MCTS + UCB) is powerful in order to **chose among alternatives**
  - By exploring "intelligently" the tree of possible consequences of potential decisions
- Works when it is possible to explore possible scenarios **by simulation**

Requires a **good model of the world**

Plan

1. Limites de l'approche classique
2. Évaluation par Monte-Carlo
3. Le compromis Exploration vs. Exploitation : algorithmes de bandits
4. Approche e-greedy
5. UCT = MCTS + UCB
6. Illustrations
7. AlaphaGo Zero

AlphaGo Zero

- Utilise MCTS pour **générer des exemples d'apprentissage** de qualité pour l'entraînement du réseau de neurones profond.
- Qui est lui-même utilisé pour générer de nouvelles parties d'Alpha Zero contre Alpha Zero.

AlphaGo Zero

- **October 2015:**

  AlphaGo wins 5-0 against a Go professional **Fan Hui**

- **March 2016:**

  AlphaGo wins 4-1 against **Lee Sedol**, winner of 18 world titles

- **January 2017:**

  An improved online version of AlphaGo, called **Master**, achieved 60 straight wins against top international players

- **May 2017:**

  Ke Jie, considered as the best human Go player, loses 3-0 against **AlphaGo** (**Master**)

- **Late 2017:**

  AlphaGo Zero is revealed and wins 100-0 against **AlphaGo**

  Self-taught using no human games

AlphaGo Zero

- **4.9 millions training games**
  vs. 30 millions training games (using human history of games) for AlphaGo
- **3 days of training**
  vs. Several months for AlphaGo
- **A single machine with 4 TPUs** (Tensor Processing Units)
  vs. Multiple machines with 48 TPUs
- Input: **the raw board description**
  vs. manually engineered descriptors of the board

AlphaGo Zero

- uses a **deep neural network** f_θ with parameters θ.
  - This neural network takes as an **input** the raw board representation s of the position and its history (7 past positions for black and 7 for white), and **outputs** both move probabilities and a value: $(p, v) = f_\theta(s)$.
  - The **vector of move probabilities** p represents the probability of selecting each move a (including pass), $p_a = \Pr(a \mid s)$.
  - The **value** v is a scalar evaluation, estimating the probability of the current player winning from position s.
  - This **neural network combines** the roles of both **policy network** and **value network** into a single architecture.

AlphaGo Zero

- Playing
  - Using UCT = MCTS + UCB
  - But no rollouts
  - Until end of game

- MCTS
  - Choses each move using $Q(s,a) + U(s,a)$ (UCB)
  - When a leaf node s' is encountered: evaluate $(P(s', \cdot), V(s')) = f_\theta(s')$
    Instead of using a rollout

AlphaGo Zero

Figure 2: **Monte-Carlo tree search in *AlphaGo Zero*.** **a** Each simulation traverses the tree by selecting the edge with maximum action-value Q, plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed). **b** The leaf node is expanded and the associated position s is evaluated by the neural network $(P(s,\cdot), V(s)) = f_\theta(s)$; the vector of P values are stored in the outgoing edges from s. **c** Action-values Q are updated to track the mean of all evaluations V in the subtree below that action. **d** Once the search is complete, search probabilities $\boldsymbol{\pi}$ are returned, proportional to $N^{1/\tau}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.

[Silver, David, et al. "***Mastering the game of go without human knowledge***. » Nature 550.7676 (2017): 354-359.]

AlphaGo Zero

Self-play in *AlphaGoZero*.

- The program plays a game s_1, ..., s_T against itself.
- In each position s_t, a Monte Carlo Tree Search (MCTS) is executed using the latest neural network f_θ.
- Moves are selected according to the search probabilities a_q computed by the MCTS, $\alpha \sim \pi_\theta$.
- The terminal position s_T is scored according to the rules of the game to compute the game winner z.

b. Neural Network Training

Learning in *AlphaGoZero*.

The neural network takes the raw board position s_t as its input, passes it through many convolutional layers with parameters θ, and outputs both a vector $\mathbf{p_t}$, representing a probability distribution over moves, and a scalar value v_t, representing the probability of the current player winning in position s_t. The neural network parameters θ are updated so as to maximise the similarity of the policy vector $\mathbf{p_t}$ to the search probabilities $\boldsymbol{\pi_t}$, and to minimise the error between the predicted winner v_t and the game winner z (see Equation 1). The new parameters are used in the next iteration of self-play **a**.

AlphaGo Zero

- AlphaGo Zero plays like humans in the **openings** and in the **end games** which seems to show that this is the way to play best

- But its **middle-game plays** are often **truly mysterious**

References

- Vincent Barra, Antoine Cornuéjols & Laurent Miclet: "*Apprentissage artificiel. Concepts et algorithms. De Bayes et Hume au Deep Learning*". Eyrolles, 2021.
- Aske Plaat: "*Deep Reinforcement Learning*". Springer, 2022
- Max Pumperla & Kevin Ferguson: "*Deep Learning and the Game of Go*". Manning, 2019
- Stuart Russell & Peter Norvig: "*Artificial Intelligence. A modern approach*". Global Edition, 2022
- Richard Sutton & Andrew Barto: "*Reinforcement Learning. An Introduction*". MIT Press, 2018 (la bible par les pionniers du RL)
- https://banditalgs.com/2016/09/18/the-upper-confidence-bound-algorithm/ (pour ceux qui aiment les maths)
- https://towardsdatascience.com/the-upper-confidence-bound-ucb-bandit-algorithm-c05c2bf4c13f (très pédagogique, avec des bouts de code python)