

# Exploration in Reinforcement Learning

## Optimism Principle and Insights from the Bandit Literature

Emilie Kaufmann & Claire Vernade



RL course, Uni. Tübingen

# The exploration issue in RL

# Outline

- 1** Introduction to Bandit Algorithms
- 2** Fixing the greedy strategy
  - A simple UCB algorithm
  - Thompson Sampling
- 3** Optimism-based heuristics for successful RL
  - (Bandit-based) Monte-Carlo Tree Search
  - From UCT to AlphaZero
  - Scalable heuristics in (Deep) RL



‘The Bandit Book’

by [Lattimore and Szepesvari, 2019]

# The Stochastic Multi-Armed Bandit Setup

$K$  arms  $\leftrightarrow K$  probability distributions :  $\nu_a$  has mean  $\mu_a$



$\nu_1$



$\nu_2$



$\nu_3$



$\nu_4$



$\nu_5$

At round  $t$ , an agent :

- ▶ chooses an arm  $A_t \in \{1, \dots, k\}$
- ▶ receives a reward  $R_t = X_{A_t, t} \sim \nu_{A_t}$

Sequential sampling strategy (bandit algorithm) :


$$p_i(F_t) = A_{t+1} = F_t(A_1, R_1, \dots, A_t, R_t).$$

Goal (for now ! ) : Maximize  $\mathbb{E} \left[ \sum_{t=1}^T R_t \right]$

# Regret of a bandit algorithm

**Bandit instance** :  $\nu = (\nu_1, \nu_2, \dots, \nu_K)$ , mean of arm  $a$  :  $\mu_a = \mathbb{E}_{X \sim \nu_a}[X]$ .

$$\mu_\star = \max_{a \in \{1, \dots, K\}} \mu_a \quad a_\star = \operatorname{argmax}_{a \in \{1, \dots, K\}} \mu_a.$$

 **best arm**

Maximizing rewards  $\leftrightarrow$  selecting  $a_\star$  as much as possible  
 $\leftrightarrow$  minimizing the **regret** [Robbins, 1952]

$$\mathcal{R}_\nu(\mathcal{A}, T) := \underbrace{T\mu_\star}_{\text{sum of rewards of an oracle strategy always selecting } a_\star} - \underbrace{\mathbb{E} \left[ \sum_{t=1}^T R_t \right]}_{\text{sum of rewards of the strategy } \mathcal{A}}$$

# Regret decomposition

$N_a(t)$  : number of selections of arm  $a$  in the first  $t$  rounds

$\Delta_a := \mu_\star - \mu_a$  : sub-optimality gap of arm  $a$

Can we think of a better (equivalent) definition of the Regret ?

$$\mathcal{R}_\nu(\mathcal{A}, T) := T\mu_\star - \mathbb{E} \left[ \sum_{t=1}^T R_t \right]$$

# Regret decomposition

## Regret decomposition

$$\mathcal{R}_\nu(\mathcal{A}, T) = \sum_{a=1}^K \Delta_a \mathbb{E}[N_a(T)].$$

**Proof.**

$$\begin{aligned}\mathcal{R}_\nu(\mathcal{A}, T) &= \mu_\star T - \mathbb{E}\left[\sum_{t=1}^T X_{A_t, t}\right] = \mu_\star T - \mathbb{E}\left[\sum_{t=1}^T \mu_{A_t}\right] \\ &= \mathbb{E}\left[\sum_{t=1}^T (\mu_\star - \mu_{A_t})\right] \\ &= \sum_{a=1}^K \underbrace{\mu_\star - \mu_a}_{\Delta_a} \mathbb{E}\left[\underbrace{\sum_{t=1}^T \mathbb{1}(A_t = a)}_{N_a(T)}\right].\end{aligned}$$



# Regret decomposition

## Regret decomposition

$$\mathcal{R}_\nu(\mathcal{A}, T) = \sum_{a=1}^K \Delta_a \mathbb{E}[N_a(T)].$$

A strategy with small regret should :

- ▶ select not too often arms for which  $\Delta_a > 0$
- ▶ ... which requires to try all arms to estimate the values of the  $\Delta_a$ 's

⇒ Exploration / Exploitation trade-off

# The greedy strategy

Select each arm once, then **exploit** the current knowledge :

$$A_{t+1} = \operatorname{argmax}_{a \in [K]} \hat{\mu}_a(t)$$

where

- ▶  $N_a(t) = \sum_{s=1}^t \mathbb{1}(A_s = a)$  is the number of selections of arm  $a$
- ▶  $\hat{\mu}_a(t) = \frac{1}{N_a(t)} \sum_{s=1}^t X_s \mathbb{1}(A_s = a)$  is the **empirical mean** of the rewards collected from arm  $a$

# The greedy strategy

Select each arm once, then **exploit** the current knowledge :

$$A_{t+1} = \operatorname{argmax}_{a \in [K]} \hat{\mu}_a(t)$$

where

- ▶  $N_a(t) = \sum_{s=1}^t \mathbb{1}(A_s = a)$  is the number of selections of arm  $a$
- ▶  $\hat{\mu}_a(t) = \frac{1}{N_a(t)} \sum_{s=1}^t X_s \mathbb{1}(A_s = a)$  is the **empirical mean** of the rewards collected from arm  $a$

**The greedy strategy can fail !**  $\nu_1 = \mathcal{B}(\mu_1), \nu_2 = \mathcal{B}(\mu_2), \mu_1 > \mu_2$

$$\mathbb{E}[N_2(T)] \geq (1 - \mu_1)\mu_2 \times (T - 1)$$

→ **Exploitation** is not enough, we need to **add some exploration**

# Is $\epsilon$ -Greedy a good idea ?

The  $\epsilon$ -greedy rule [Sutton and Barto, 1998] is a simple randomized way to alternate exploration and exploitation.

## $\epsilon$ -greedy strategy

At round  $t$ ,

- ▶ with probability  $\epsilon$

$$A_t \sim \mathcal{U}(\{1, \dots, K\})$$

- ▶ with probability  $1 - \epsilon$

$$A_t = \operatorname{argmax}_{a=1, \dots, K} \hat{\mu}_a(t).$$

# Is $\epsilon$ -Greedy a good idea ?

The  $\epsilon$ -greedy rule [Sutton and Barto, 1998] is a simple randomized way to alternate exploration and exploitation.

## $\epsilon$ -greedy strategy

At round  $t$ ,

- ▶ with probability  $\epsilon$

$$A_t \sim \mathcal{U}(\{1, \dots, K\})$$

- ▶ with probability  $1 - \epsilon$

$$A_t = \operatorname{argmax}_{a=1, \dots, K} \hat{\mu}_a(t).$$

→ Linear regret :  $\mathcal{R}_\nu(\epsilon\text{-greedy}, T) \geq \epsilon \frac{K-1}{K} \Delta_{\min} T.$

$$\Delta_{\min} = \min_{a: \mu_a < \mu_*} \Delta_a$$

# Is $\epsilon$ -Greedy a good idea ?

Can we tune  $\epsilon$  or find a good schedule  $\epsilon_1, \dots, \epsilon_T$  ?

# Is $\epsilon$ -Greedy a good idea ?

Can we tune  $\epsilon$  or find a good schedule  $\epsilon_1, \dots, \epsilon_T$  ?

**Answer :** Yes but... one must choose  $\epsilon_t = \min \left( 1, \frac{K}{\Delta_{\min}^2 t} \right)$  [] Auer et al., 2002

Why is it an issue ?

# Is $\epsilon$ -Greedy a good idea ?

Can we tune  $\epsilon$  or find a good schedule  $\epsilon_1, \dots, \epsilon_T$  ?

**Answer :** Yes but... one must choose  $\epsilon_t = \min \left( 1, \frac{K}{\Delta_{\min}^2 t} \right)$   $\square$

Why is it an issue ?

**Answer :** We do not know  $\Delta_{\min}$  !

Conclusions :

- ▶  $\epsilon$ -Greedy has linear regret (it always explores !)
- ▶ It can only be fixed in theory, not in practice (try it out !)
- ▶ Alternative : Explore-Then-Commit (ETC, see tutorial)
- ▶ Better : The Optimism Principle



# The optimism principle

**Step 1** : construct a set of statistically plausible models

- For each arm  $a$ , build a confidence interval on the mean  $\mu_a$  :

$$\mathcal{I}_a(t) = [\text{LCB}_a(t), \text{UCB}_a(t)]$$

LCB = Lower Confidence Bound

UCB = Upper Confidence Bound

$$P(\mu \in \mathcal{I}_a(t)) > 1 - \delta$$

$\delta \rightarrow 0$

- **Goal** : Find LCB and UCB such that

$$\mathbb{P}(\text{LCB}_a(t) < \mu_a < \text{UCB}_a(t)) \approx 1$$

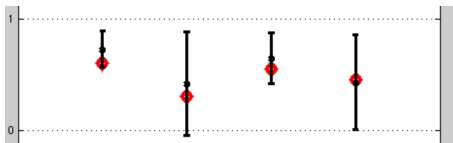


FIGURE – Confidence intervals on the means after  $t$  rounds

# The optimism principle

**Step 2** : act as if the best possible model were the true model  
(*optimism in face of uncertainty*)

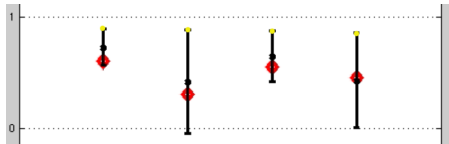


FIGURE – Confidence intervals on the means after  $t$  rounds

► That is, select

$$A_{t+1} = \operatorname{argmax}_{a=1,\dots,K} \text{UCB}_a(t).$$

# Outline

- 1 Introduction to Bandit Algorithms
- 2 Fixing the greedy strategy
  - A simple UCB algorithm
  - Thompson Sampling
- 3 Optimism-based heuristics for successful RL
  - (Bandit-based) Monte-Carlo Tree Search
  - From UCT to AlphaZero
  - Scalable heuristics in (Deep) RL

# How to build confidence intervals ?

We need  $UCB_a(t)$  such that for a small  $\delta > 0$ ,

$$\mathbb{P}(\mu_a \leq UCB_a(t)) > 1 - \delta.$$

→ tool : concentration inequalities

**Example** : rewards are  $\sigma^2$  sub-Gaussian

## Hoeffding inequality

$Z_i$  i.i.d. with mean  $\mu$ . For all  $s \geq 1$

$$\mathbb{P}\left(\underbrace{\hat{\mu}_s}_\text{UCB} + x < \mu\right) \leq \exp\left(-\frac{sx^2}{2\sigma^2}\right)$$

**Exercise** : Tune  $x$  such that  $\delta = \exp\left(-\frac{sx^2}{2\sigma^2}\right)$ .

# How to build confidence intervals ?

We need  $\text{UCB}_a(t)$  such that for a small  $\delta > 0$ ,

$$\mathbb{P}(\mu_a \leq \text{UCB}_a(t)) > 1 - \delta.$$

→ tool : concentration inequalities

**Example** : rewards are  $\sigma^2$  sub-Gaussian

## Hoeffding inequality

$Z_i$  i.i.d. with mean  $\mu$ . For all  $s \geq 1$

$$\mathbb{P}\left(\underbrace{\hat{\mu}_s}_\text{UCB} + x < \mu\right) \leq \exp\left(-\frac{sx^2}{2\sigma^2}\right)$$

**Exercise** : Tune  $x$  such that  $\delta = \exp\left(-\frac{sx^2}{2\sigma^2}\right)$ .

# How to build confidence intervals ?

**Goal :** Choose a sequence of confidence levels  $(\delta_t)_{t=1\dots\infty}$  such that

$$\sum_{t=1}^{\infty} \mathbb{P} \left( \underbrace{\mu_a > \hat{\mu}_a(t) + \sqrt{\frac{2\sigma^2 \log(1/\delta_t)}{N_a(t)}}}_{UCB_a(t)} \right) \leq O(1)$$

- ▶  $N_a(t) = \sum_{s=1}^t \mathbb{1}_{(A_s=a)}$  number of selections of  $a$  after  $t$  rounds
- ▶  $\hat{\mu}_{a,s} = \frac{1}{s} \sum_{k=1}^s Y_{a,k}$  average of the first  $s$  observations from arm  $a$
- ▶  $\hat{\mu}_a(t) = \hat{\mu}_{a,N_a(t)}$  empirical estimate of  $\mu_a$  after  $t$  rounds

How would you choose the sequence of confidence levels  $\delta_t$  ?

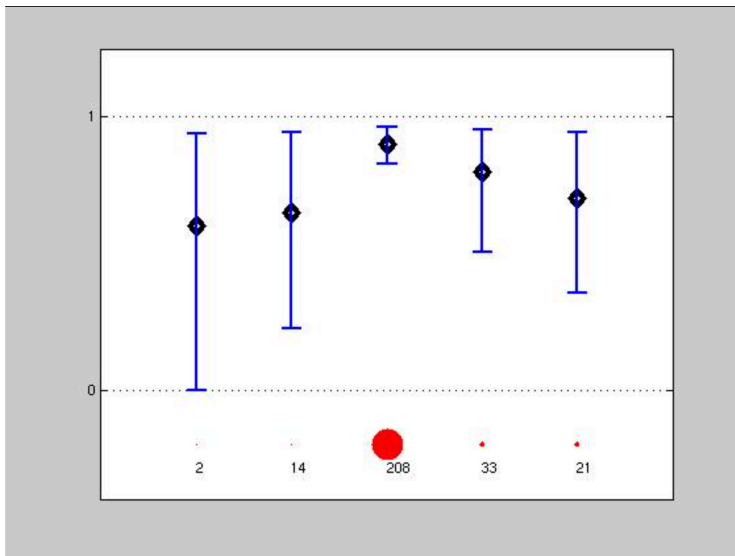
# A first UCB algorithm

UCB( $\alpha$ ) selects  $A_{t+1} = \operatorname{argmax}_a \text{UCB}_a(t)$  where

$$\text{UCB}_a(t) = \underbrace{\hat{\mu}_a(t)}_{\text{exploitation term}} + \underbrace{\sqrt{\frac{\alpha \log(t)}{N_a(t)}}}_{\text{exploration bonus}}.$$

- ▶ this form of UCB was first proposed for Gaussian rewards [Katehakis and Robbins, 1995]
- ▶ popularized by [Auer et al., 2002] for bounded rewards : UCB1, for  $\alpha = 2\sigma^2$
- ▶ the analysis of UCB( $\alpha$ ) was further refined to hold for  $\alpha > 1/2$  in that case [Bubeck, 2010, Cappé et al., 2013]

# UCB in action





# Regret bound for UCB( $\alpha$ )

## Theorem

For  $\sigma^2$ -subGaussian rewards, the UCB algorithm with parameter  $\alpha = 6\sigma^2$  satisfies, for any sub-optimal arm  $a$ ,

$$\mathbb{E}_{\mu}[N_a(T)] \leq \frac{24\sigma^2}{\Delta_a^2} \log(T) + 1 + \frac{\pi^2}{3}$$

where  $\Delta_a = \mu_{\star} - \mu_a$ .

## Consequence :

$$\mathcal{R}_{\nu}(\text{UCB}(6\sigma^2), T) \leq \left( \sum_{a: \mu_a < \mu_{\star}} \frac{24\sigma^2}{\Delta_a^2} \right) \log(T) + \left( 1 + \frac{\pi^2}{3} \right) \sum_{a=1}^K \Delta_a$$

# Bayesian alternative : Thompson Sampling

Recall Bayes' rule :

$$\mathbb{P}(\mu|X_1, \dots, X_n) = \frac{\mathbb{P}(X_1, \dots, X_n|\mu)\mathbb{P}_0(\mu)}{\int \mathbb{P}(X_1, \dots, X_n|\mu)\mathbb{P}_0(\mu)d\mu}$$

where

$\mathbb{P}_0$  is a **prior** distribution over  $\mu$ , (ex : Beta  $B(\mu; \alpha, \beta) \propto \mu^{\alpha-1}(1-\mu)^{\beta-1}$ )

$\mathbb{P}(X_1, \dots, X_n|\mu)$  is the likelihood of the observations under  $\mu$

(ex : Bernoulli  $\text{Ber}(x; \mu) \propto \mu^x(1-\mu)^{1-x}$ )

**Exercise :** Given a Beta prior and  $n$  Bernoulli observations, compute the posterior distribution over  $\mu$

# Bayesian alternative : Thompson Sampling

Recall Bayes' rule :

$$\mathbb{P}(\mu|X_1, \dots, X_n) = \frac{\mathbb{P}(X_1, \dots, X_n|\mu)\mathbb{P}_0(\mu)}{\int \mathbb{P}(X_1, \dots, X_n|\mu)\mathbb{P}_0(\mu)d\mu}$$

where

$\mathbb{P}_0$  is a **prior** distribution over  $\mu$ , (ex : Beta  $B(\mu; \alpha, \beta) \propto \mu^{\alpha-1}(1-\mu)^{\beta-1}$ )


$\mathbb{P}(X_1, \dots, X_n|\mu)$  is the likelihood of the observations under  $\mu$

(ex : Bernoulli  $\text{Ber}(x; \mu) \propto \mu^x(1-\mu)^{1-x}$ )

**Exercise :** Given a Beta prior and  $n$  Bernoulli observations, compute the posterior distribution over  $\mu$

At round  $t$ , denote  $\mathbb{P}_t(\mu_a) = \mathbb{P}(\mu_a|X_1, \dots, X_t)$ , for each arm  $a$ .

**Thompson Sampling :**

- ▶ Sample  $\tilde{\mu}_a \sim \mathbb{P}_t(\mu_a)$  *hallucinated arm parameter*
- ▶ Choose  $A_{t+1} = \operatorname{argmax}_a \tilde{\mu}_a$  *best arm under posterior*
- ▶ Update  $\mathbb{P}_{t+1}(\mu_{A_{t+1}}) \propto \mathbb{P}(R_{t+1}|\mu_a)\mathbb{P}_t(\mu_a)$  

# Two principles for exploration

## Optimism (UCB)

### ► Assumptions :

- Noise is sub-Gaussian or bounded
- Independent arms

### ► Action selection rule :

- Select arm  $a_t = \arg \max_a \left( \hat{\mu}_a + \sqrt{\frac{\alpha \log t}{N_a}} \right)$
- $\hat{\mu}_a$  is the empirical mean reward for arm  $a$
- $N_a$  is the number of times arm  $a$  has been chosen

### ► Update rule :

- Update  $\hat{\mu}_a$  and  $N_a$  after observing reward

## Thompson Sampling

### ► Assumptions :

- Noise follows an exact probabilistic model
- Preferably noise model from the exponential family

### ► Action selection rule :

- Sample  $\theta_a$  from the posterior distribution for each arm  $a$
- Select arm  $a_t = \arg \max_a \theta_a$

### ► Update rule :

- Update posterior distribution for each arm based on observed reward

# Conclusions and Take-Home message

- ▶ **Undirected exploration** is inefficient and needs knowledge of the reward to be tuned properly ;
- ▶ **The optimism principle** says that, given high-probability confidence bound, one should act according to the **largest plausible outcome predicts** ;
- ▶ The Upper Confidence Bound algorithm (UCB) achieves logarithmic regret and simply adds an exploration bonus proportional to  $1/\sqrt{N_a(t)}$  :

$$A_{t+1} = \operatorname{argmax}_a \hat{\mu}_a(t) + \sqrt{\frac{\alpha \log(t)}{N_a(t)}}$$

- ▶ Thompson Sampling offers a great (Bayesian) alternative (try it out !)

# Outline

- 1 Introduction to Bandit Algorithms
- 2 Fixing the greedy strategy
  - A simple UCB algorithm
  - Thompson Sampling
- 3 Optimism-based heuristics for successful RL
  - (Bandit-based) Monte-Carlo Tree Search
  - From UCT to AlphaZero
  - Scalable heuristics in (Deep) RL

# RL on Trees

- ▶ Trees are used to model large combinatorial games (nodes represent the state after each move).
- ▶ Searching trees is a hard non-convex optimization problem.
- ▶ It can be formulated as an RL problem : find action (at root) to maximize long-term pay-off (leaf of the tree)
- ▶ Need to **estimate** return through **simulations**
- ▶ Need to **account for uncertainty**

**AlphaGo** and **AlphaZero** are examples of (neural) Upper Confidence on Trees (UCT) algorithms.

**Beyond games**, MCTS algorithms had impact on many Science and Engineering domains [Kemmerling et al., 2024].

# UCB for Trees : 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  $s_1$** .

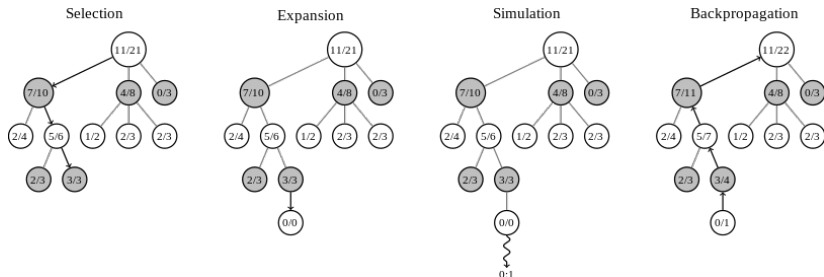


FIGURE – A generic MCTS algorithm for a game

MCTS requires a generative model (to sample trajectories from  $s_1$ )



# The UCT algorithm

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

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

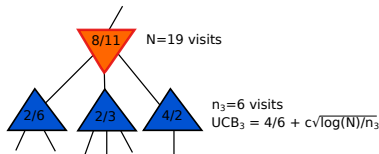
## UCT in a Game Tree

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

$$\operatorname{argmax}_{a \in C(s)} \frac{S(s, a)}{N(s, a)} + c \sqrt{\frac{\log(\sum_b N(s, b))}{N(s, a)}}$$

$N(s, a)$  : 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 child (next action) needs to be selected

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

## UCT in a Game Tree

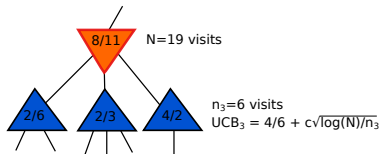
In a **MIN node**  $s$  (= adversary move), select an action

$$\operatorname{argmin}_{a \in C(s)} \frac{S(s, a)}{N(s, a)} - c \sqrt{\frac{\log(\sum_b N(s, b))}{N(s, a)}}$$

Use LCB to simulate adversary's move

$N(s, a)$  : number of visits of  $(s, a)$

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



# The UCT algorithm

For each move in the game, fix a number of playouts or a search depth and perform **Selection, Expansion, Backpropagation** until a reasonable estimate of the best next action is found.

**Output of planning algorithm :** At the root  $s_0$  of the tree, several option :

- ▶ Greedy :  $\hat{a} = \operatorname{argmax}_a \frac{S(s_0, a)}{N(s_0, a)} ;$
- ▶ UCB (Optimistic) :  $\hat{a} = \operatorname{argmax}_a \frac{S(s_0, a)}{N(s_0, a)} + c \sqrt{\frac{\log(\sum_b N(s, b))}{N(s, a)}} ;$
- ▶ Randomized : Sample  $\hat{a} \sim \mathcal{D}(\hat{\mu}_1, \dots, \hat{\mu}_K) ;$
- ▶ Other heuristics...

# The UCT algorithm

For each move in the game, fix a number of playouts or a search depth and perform **Selection, Expansion, Backpropagation** until a reasonable estimate of the best next action is found.

**Output of planning algorithm :** At the root  $s_0$  of the tree, several option :

- ▶ Greedy :  $\hat{a} = \operatorname{argmax}_a \frac{S(s_0, a)}{N(s_0, a)} ;$
- ▶ UCB (Optimistic) :  $\hat{a} = \operatorname{argmax}_a \frac{S(s_0, a)}{N(s_0, a)} + c \sqrt{\frac{\log(\sum_b N(s, b))}{N(s, a)}} ;$
- ▶ Randomized : Sample  $\hat{a} \sim \mathcal{D}(\hat{\mu}_1, \dots, \hat{\mu}_K) ;$
- ▶ Other heuristics...

Remarks :

- ▶ 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

- 1 Introduction to Bandit Algorithms
- 2 Fixing the greedy strategy
  - A simple UCB algorithm
  - Thompson Sampling
- 3 Optimism-based heuristics for successful RL
  - (Bandit-based) Monte-Carlo Tree Search
  - From UCT to AlphaZero
  - Scalable heuristics in (Deep) RL

# AlphaGo idea

AlphaGo learns a good policy by using a MCTS algorithm **guided by a neural network**

≠ pure play-out based MCTS

## Input

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

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

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

# AlphaGo idea

AlphaGo learns a good policy by using a MCTS algorithm **guided by a neural network**

≠ pure play-out based MCTS

## Input

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

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

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

for some (fine-tuned) constant  $c$ .

# AlphaGo idea

AlphaGo learns a good policy by using a MCTS algorithm **guided by a neural network**

≠ pure play-out based MCTS

## Input

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

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  $(\mathbf{p}, v) = f_{\theta}(s_L)$ .

- ▶ 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  $P(s_L, b) = \mathbf{p}_b$  (possibly add some noise)



# AlphaGo idea

AlphaGo learns a good policy by using a MCTS algorithm **guided by a neural network**

≠ pure play-out based MCTS

## Input

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

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  $s_L$ ,

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$$

$$S(s_t, a_t) \leftarrow S(s_t, a_t) + v$$

# AlphaGo idea

AlphaGo learns a good policy by using a MCTS algorithm **guided by a neural network**

≠ pure play-out based MCTS

## Input

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

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  $\tau$ .

# Training the neural network

- ▶ In AlphaGo,  $f_\theta$  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  $\theta$  be the current parameter of the network  $(\mathbf{p}, v) = f_\theta(s_L)$ .

- 1 generate  $N$  games where each player uses MCTS( $\theta$ ) to select the next action  $a_t$  (and output a probability over actions  $\pi_t$ )

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

$T_i$  : 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

$$L(s, \pi, z; \mathbf{p}, v) = (z - v)^2 - \pi^\top \log(\mathbf{p}) + c_v \|v\|^2 + c_p \|\mathbf{p}\|^2$$

# A nice actor-critic architecture

AlphaZero alternates between

- ▶ **The actor** :  $\text{MCTS}(\theta)$   
generates trajectories guided by the network  $f_\theta$  but still exploring
- act as a **policy improvement**  
( $N = 25000$  games played, in which the choice of each move uses MCTS with 1600 simulations)
- ▶ **The critic** : neural network  $f_\theta$   
updates  $\theta$  based on trajectories followed by the critic
- **evaluate** the actor's policy

# Outline

- 1 Introduction to Bandit Algorithms
- 2 Fixing the greedy strategy
  - A simple UCB algorithm
  - Thompson Sampling
- 3 Optimism-based heuristics for successful RL**
  - (Bandit-based) Monte-Carlo Tree Search
  - From UCT to AlphaZero
  - Scalable heuristics in (Deep) RL

# Optimism and Thompson Sampling in RL

- ▶ We can generalize the Optimism Principle to MDPs :

*Find the largest possible **value**  $\max_a \bar{Q}(s, a)$  in the best possible MDP*

# Optimism and Thompson Sampling in RL

- ▶ We can generalize the Optimism Principle to MDPs :

*Find the largest possible **value**  $\max_a \bar{Q}(s, a)$  in the best possible MDP*

- ▶ Similarly, Thompson Sampling becomes **Posterior Sampling**.  
[Osband et al., 2013]

*Sample a plausible MDP under the posterior on the parameters, and find optimal policy by e.g. Dynamic Programming.*

# Limitations of optimistic approaches

An important message from optimistic approaches :

- Do not only trust the estimated MDP  $\hat{M}_t$ , but take into account the **uncertainty** in the underlying estimate

UCB-VI : find best policy in best plausible MDP  $M_t^+$  :

$$\begin{aligned}\pi_{t,h}^+(s) &= \operatorname{argmax}_{a \in \mathcal{A}} Q_h^{*, M_t^+}(s, a) \\ &= \operatorname{argmax}_{a \in \mathcal{A}} \max_{M \in \mathcal{M}_t} Q_h^{*, M}(s, a)\end{aligned}$$



# Limitations of optimistic approaches

An important message from optimistic approaches :

- Do not only trust the estimated MDP  $\hat{M}_t$ , but take into account the **uncertainty** in the underlying estimate

UCB-VI : find best policy in best plausible MDP  $M_t^+$  :

$$\begin{aligned}\pi_{t,h}^+(s) &= \operatorname{argmax}_{a \in \mathcal{A}} Q_h^{*, M_t^+}(s, a) \\ &= \operatorname{argmax}_{a \in \mathcal{A}} \max_{M \in \mathcal{M}_t} Q_h^{*, M}(s, a)\end{aligned}$$

## Scaling for large state-action spaces ?

- ▶ each state action pair may be visited only very little...
- ▶ UCB-VI is quite inefficient in practice for very large state-spaces

# Extrinsic versus intrinsic reward

UCB-VI is a model-based approach.

- ▶ a pure model-based algorithm would estimate the model  $(\hat{r}, \hat{p})$  and play the optimal policy in the estimated model
- ▶ instead, it plays the optimal policy in a modified model where the reward is replaced by

$$\hat{r}_h^t(s, a) + \underbrace{\beta_t(s, a)}_{\substack{\text{add some reward for the} \\ \text{visitation of undervisited states} \\ = \text{intrinsic reward}}}$$

**More general idea** : run any (possibly model-free) RL algorithm replacing the collected (extrinsic) reward  $r_t$  by

$$r_t^+ = r_t + r_t^I$$

where  $r_t^I$  is *some* form of intrinsic reward.

# Count-based exploration

## General principle

- ➊ Estimate a “proxi” for the number of visits of a state  $\tilde{n}_t(s)$
- ➋ Add an exploration bonus directly to the collected rewards :

$$r_t^+ = r_t + c \sqrt{\frac{1}{\tilde{n}_t(s_t)}}$$

- ➌ Run any (Deep)RL algorithm on  $\mathcal{D} = \bigcup_t \{(s_t, a_t, r_t^+, s_{t+1})\}$ .

# Count-based exploration

## General principle

- 1 Estimate a “proxi” for the number of visits of a state  $\tilde{n}_t(s)$
- 2 Add an exploration bonus directly to the collected rewards :

$$r_t^+ = r_t + c \sqrt{\frac{1}{\tilde{n}_t(s_t)}}$$

- 3 Run any (Deep)RL algorithm on  $\mathcal{D} = \bigcup_t \{(s_t, a_t, r_t^+, s_{t+1})\}$ .

## Example of pseudo-counts :

- ▶ use **density estimation** [Bellemare et al., 2016]
- ▶ use a **hash function**, e.g.  $\phi : \mathcal{S} \rightarrow \{-1, 1\}^k$  [Tang et al., 2017]  
 $n(\phi(s_t)) \leftarrow n(\phi(s_t)) + 1$
- ▶ use **kernels**  $n(s_t) = \sum_{s \in \mathcal{H}_t} K(s_t, s)$  [Badia et al., 2020]

# Limitations of Posterior Sampling

An important message from posterior sampling :

→ Adding some noise to the estimated MDP  $\hat{M}_t$  is helpful !

$$\begin{aligned}\tilde{r}_t(s, a) &= \hat{r}_t(s, a) + \epsilon_t(s, a) \\ \tilde{p}_t(s'|s, a) &= \hat{p}_t(\cdot|s, a) + \epsilon'_t(s, a).\end{aligned}$$

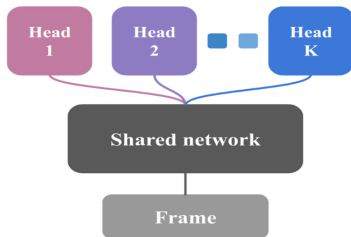
## Scaling for large state-action spaces ?

- ▶ maintaining independent posterior over all state action rewards and transitions can be costly
- ▶ more sophisticated prior distributions encoding some structure and the associated posteriors can be hard to sample from

→ use other type of randomized exploration, not necessarily Bayesian

# Bootstrap DQN

**Idea** : maintain a “distribution of Q-values”  
( $\neq$  model-based method)



$K$  “bootstrap” heads to generate  $K$  different estimated Q-values  
 $Q_1(s, a), \dots, Q_K(s, a)$

[Osband et al., 2016]

# Bootstrap DQN

---

**Algorithm 1** Bootstrapped DQN

---

```
1: Input: Value function networks  $Q$  with  $K$  outputs  $\{Q_k\}_{k=1}^K$ . Masking distribution  $M$ .  
2: Let  $B$  be a replay buffer storing experience for training.  
3: for each episode do  
4:   Obtain initial state from environment  $s_0$   
5:   Pick a value function to act using  $k \sim \text{Uniform}\{1, \dots, K\}$   
6:   for step  $t = 1, \dots$  until end of episode do  
7:     Pick an action according to  $a_t \in \arg \max_a Q_k(s_t, a)$   
8:     Receive state  $s_{t+1}$  and reward  $r_t$  from environment, having taking action  $a_t$   
9:     Sample bootstrap mask  $m_t \sim M$   
10:    Add  $(s_t, a_t, r_{t+1}, s_{t+1}, m_t)$  to replay buffer  $B$   
11:  end for  
12: end for
```

---

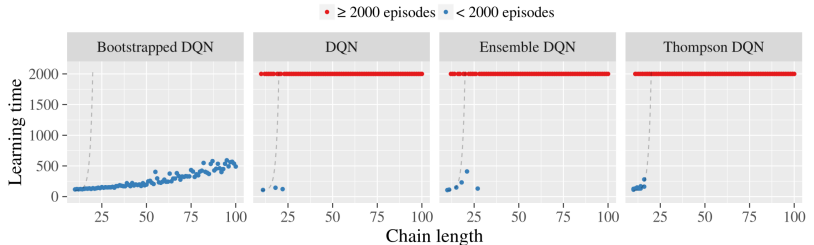
$m_t \in \{0, 1\}^k$  determines which heads will be updated with the gradient

$$g_t^k = m_t^k \left( r_t + \gamma \max_b Q(s_t, b; \theta_-) - Q_k(s_t, a_t; \theta) \right) \nabla_{\theta} Q_k(s_t, a_t; \theta)$$

# Bootstrap DQN



“Deep Exploration” occurs with Bootstrap DQN



**Another possible idea :**

→ add noise in the neural network parameters

(e.g. Noisy Networks [Fortunato et al., 2017])



# Conclusion : Bandits for RL

Bandits tools are useful for Reinforcement Learning :

- ▶ UCB-VI, PSRL : bandit-based exploration for tabular MDPs
- ▶ ... that can motivate “deeper” heuristics

Bandit tools led to big success in Monte-Carlo planning

- ▶ ... without proper sample complexity guarantees
- ➔ Unifying theory and practice is a big challenge in RL !



Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002).  
Finite-time analysis of the multiarmed bandit problem.  
*Machine Learning*, 47(2) :235–256.



Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., and Blundell, C. (2020).  
Never give up : Learning directed exploration strategies.  
In *ICLR*.



Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016).  
Unifying count-based exploration and intrinsic motivation.  
In *Advances in Neural Information Processing Systems (NIPS)*.



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.



Bubeck, S. (2010).  
*Jeux de bandits et fondation du clustering*.  
PhD thesis, Université de Lille 1.



Cappé, O., Garivier, A., Maillard, O.-A., Munos, R., and Stoltz, G. (2013).  
Kullback-Leibler upper confidence bounds for optimal sequential allocation.  
*Annals of Statistics*, 41(3) :1516–1541.



Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. (2017).  
Noisy networks for exploration.  
*arXiv :1706.10295*.



Katehakis, M. and Robbins, H. (1995).  
Sequential choice from several populations.  
*Proceedings of the National Academy of Science*, 92 :8584–8585.



Kemmerling, M., Lütticke, D., and Schmitt, R. H. (2024).  
Beyond games : a systematic review of neural monte carlo tree search applications.  
*Applied Intelligence*, 54(1) :1020–1046.



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.



Lattimore, T. and Szepesvari, C. (2019).

*Bandit Algorithms.*

Cambridge University Press.



Osband, I., Blundell, C., Pritzel, A., and Roy, B. V. (2016).

Deep exploration via bootstrapped DQN.

In *Advances in Neural Information Processing Systems (NIPS)*.



Osband, I., Van Roy, B., and Russo, D. (2013).

(More) Efficient Reinforcement Learning Via Posterior Sampling.

In *Advances in Neural Information Processing Systems*.



Robbins, H. (1952).

Some aspects of the sequential design of experiments.

*Bulletin of the American Mathematical Society*, 58(5) :527–535.



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–.



Sutton, R. and Barto, A. (1998).

*Reinforcement Learning : an Introduction*.

MIT press.



Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., Turck, F. D., and Abbeel, P. (2017).

#exploration : A study of count-based exploration for deep reinforcement learning.

In *Advances in Neural Information Processing Systems (NIPS)*.