# Introduction to Reinforcement Learning

A mini course @ HCMUS, Vietnam

Lectures 7-9 (cont'd)

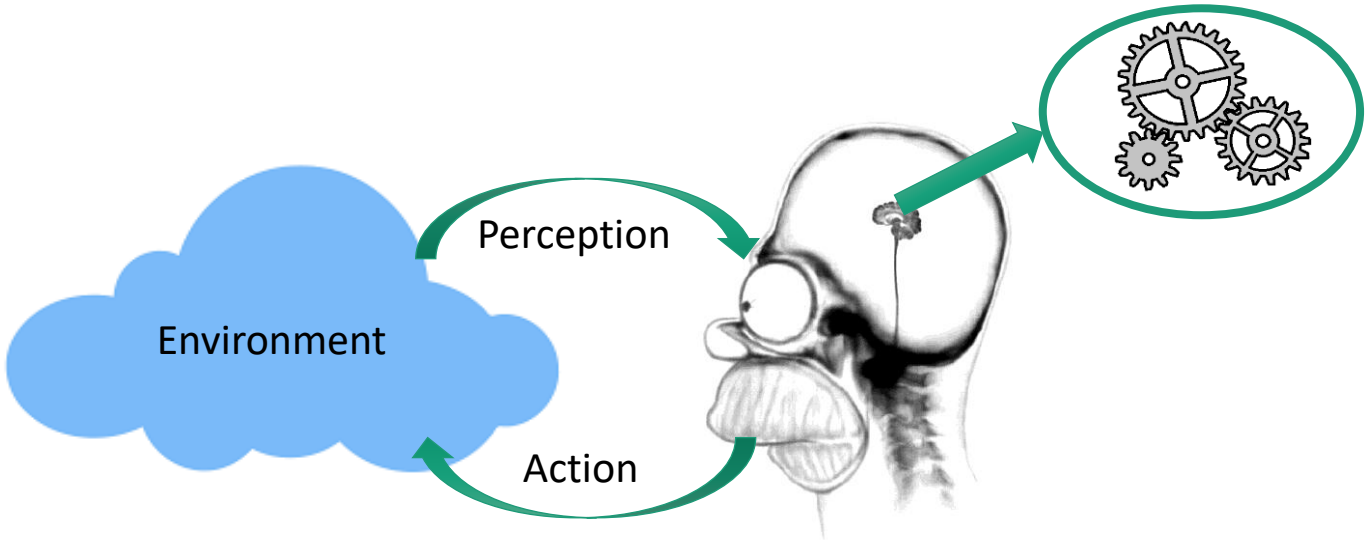
Long Tran-Thanh

long.tran-thanh@warwick.ac.uk

University of Warwick, UK

# Multi-Armed Bandits

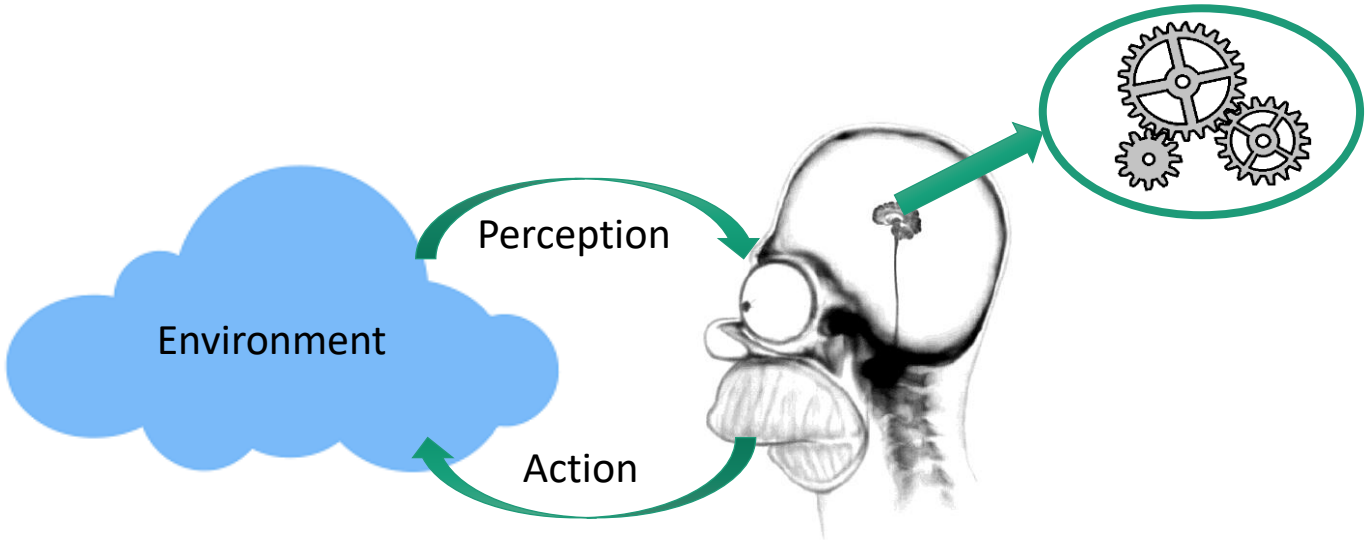
# Optimisation problems in sequential decision making




Until now, we focused more on the learning problem in SDM:

- Learn the value function
- Learn the optimal policy

# Optimisation problems in sequential decision making

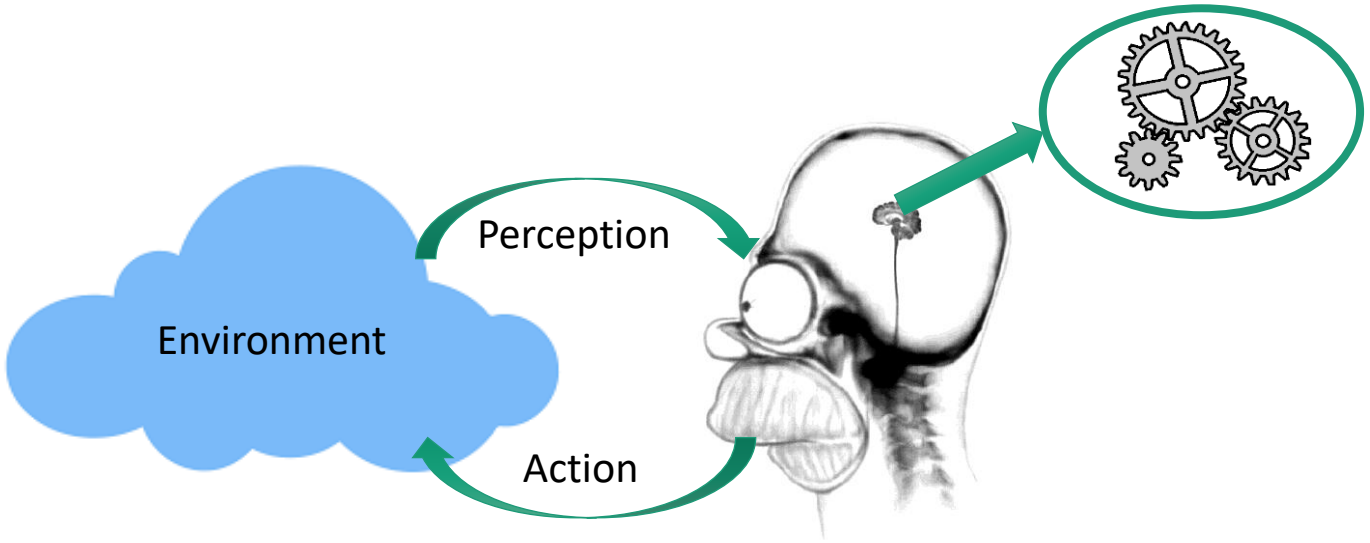



Until now, we focused more on the learning problem in SDM:

- Learn the value function
- Learn the optimal policy

Example: episodic MDP -> learn the model as accurately as possible, given a number of episodes

# Optimisation problems in sequential decision making

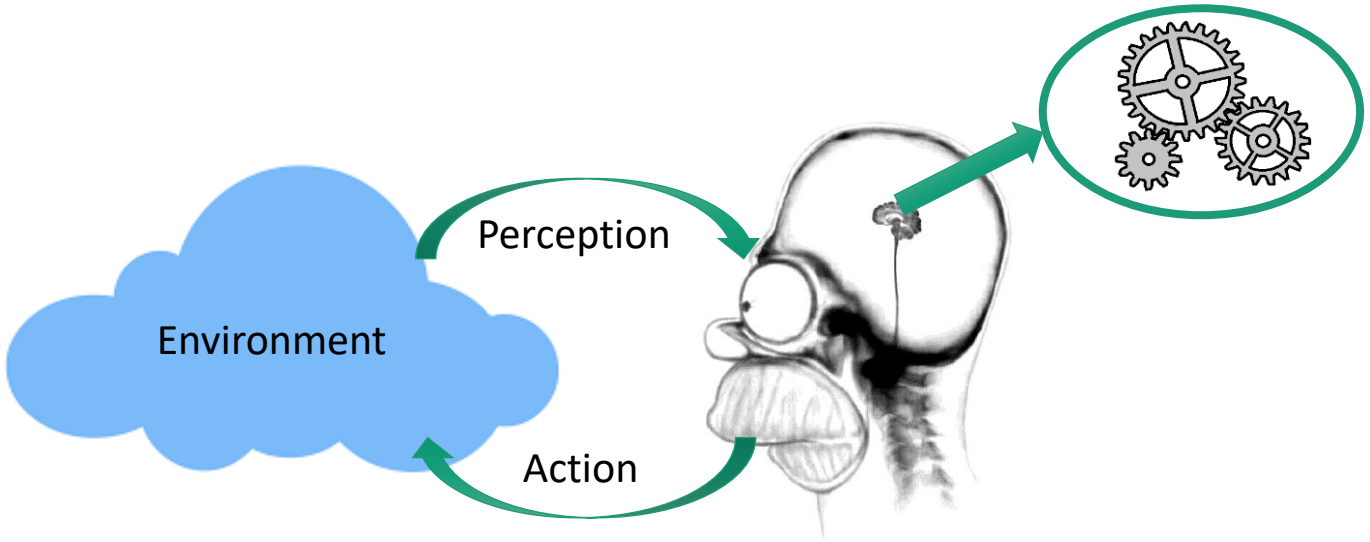



Until now, we focused more on the learning problem in SDM:

- Learn the value function
- Learn the optimal policy

Question: What will we do if we need to find the optimal policy as quickly as possible, and then use it in the rest of the remaining episodes? (i.e., we need to perform at the same time while we are learning as well?)

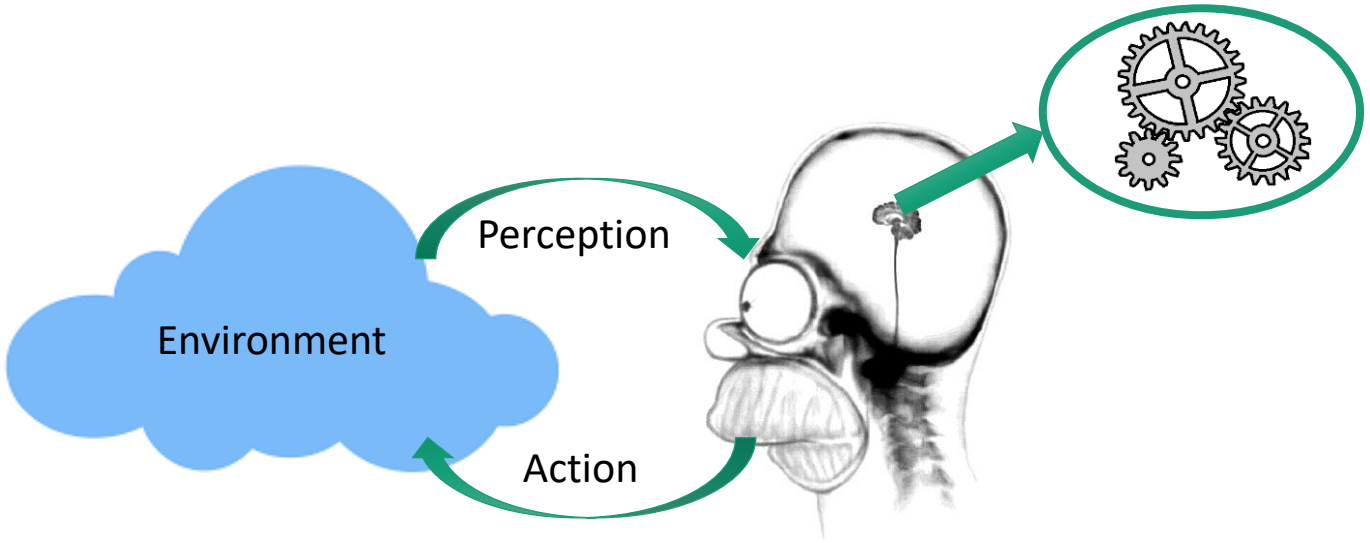
# Optimisation problems in sequential decision making




Put differently:

- Learning: Given T episodes, our goal is to find the best policy within this T episodes
- Optimisation: Given T episodes, each delivers a return. We need to **maximise our total return over T episodes**

# Optimisation problems in sequential decision making




Put differently:

- Learning: Given T episodes, our goal is to find the best policy within this T episodes
- Optimisation: Given T episodes, each delivers a return. We need to **maximise our total return over T episodes**

New challenge: exploration vs. exploitation

# Exploration vs. Exploitation

Trade-off

**Exploration**

(learn the reward values)

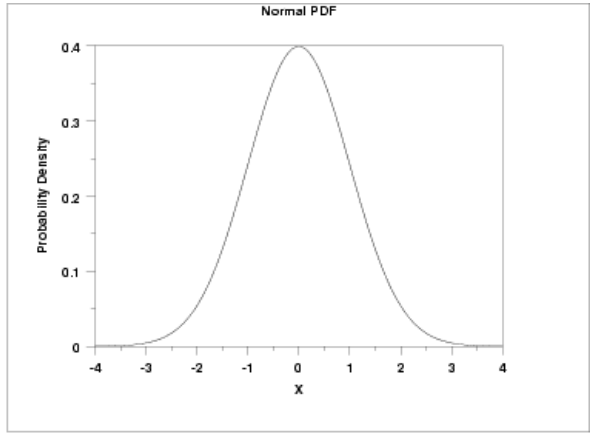
**Exploitation**

(optimise the rewards over time)

Too much exploitation: not enough information -> suboptimality

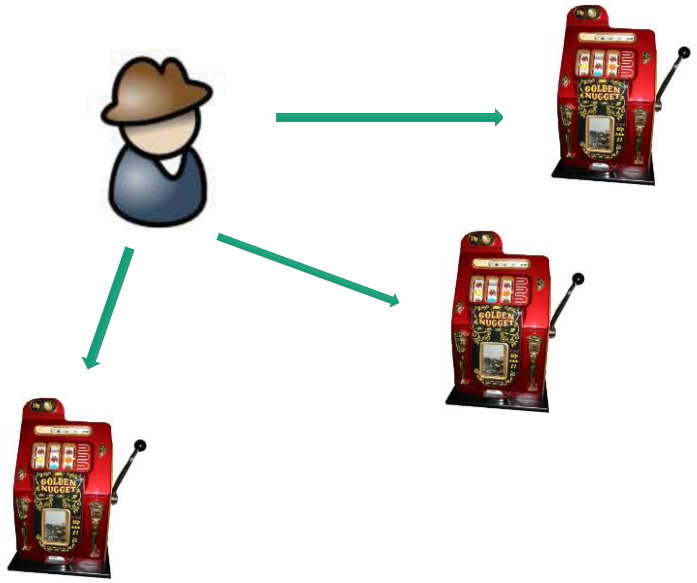
Too much exploration: not enough time to optimise -> suboptimality

# One-armed bandit




Reward value is drawn from unknown distribution

# The multi-armed bandit (MAB) model



There are multiple arms

At each time step (round):

- We choose 1 arm to pull
- Receive a reward, drawn from an unknown distribution of that arm

Objective: maximise the expected total reward

Exploration: we want to learn each arm's expected reward value

Exploitation: we want to maximise the sum of the rewards

MAB is the **simplest model** that captures the dilemma of exploration vs. exploitation

# How to solve MAB problems?

- We don’t have knowledge about the expected reward values at the beginning.
- But we can learn these values through exploration.
- However, this indicates that we have to pull arms that are not optimal (to learn that they are not optimal).
- We cannot achieve the optimal solution, which would be to pull the optimal arm (the one with the highest expected reward value) all the time.
- Our goal: design algorithms that are close to the optimum as much as possible (= good approximation)

# Performance of bandit algorithms

So if we cannot achieve the best possible, then what else can we do?

Answer: we can aim to design algorithms that have close performance to that of the best possible

A performance measurement metric: regret

Regret: the difference between the (expected) performance of an algorithm with that of the best possible

# No-regret algorithms

Regret is defined over a finite number of time steps (in the example, it was 100 time step).

Average regret: = regret divided by the number of time steps (i.e., it measures how much is our regret on average at each time step)

No-regret algorithm: = if the average regret of the algorithm converges to 0 as the size of horizon (i.e., number of time steps) goes to infinity

Example: an algorithm with the following regrets

| Time horizon size (T) | 100 | 200 | 300 | … | 10,000 |
|---|---|---|---|---|---|
| Regret (R) | 10 | 14.1 | 17.3 | | 100 |

In general, the algorithm has regret R = $\sqrt{T}$ where $T$ is the time horizon

The average regret in this case is $\frac{\sqrt{T}}{T} = \frac{1}{\sqrt{T}}$ for each time horizon $T$ -> no-regret (as it converges to 0)

# The UCB algorithm for MABs

Assumption: all the rewards are between 0 and C

Initial phase: pull each arm once

Main phase:

$i(t)$ : the arm we pull at time step *t*

$n_i(t)$ : the number of times we have pulled arm *I* up to *t*

$\hat{r}_i(t) = \frac{\sum_{\tau=1}^{t} r_i(\tau) I\{i(t) = i\}}{n_i(t)}$ : average reward of arm *i* at *t*

UCB index of arm i: $\hat{r}_i(t) + C\sqrt{\frac{2 \ln t}{n_i(t)}}$

Pulling policy: pull the arm with highest UCB index (*= i(t)*)

Update knowledge (number of pulls, average reward) of *i(t)*

# A useful lemma: Hoeffding's inequality

$$\mu = \mathbb{E}[X]$$

$$\hat{X}_N = \frac{\sum_{i=1}^{N} x_i}{N}$$

Hoeffding's inequality:

$$P(\hat{x}_N > \mu + \varepsilon) \leq e^{-2N\varepsilon^2}$$

$$P(\hat{x}_N < \mu - \varepsilon) \leq e^{-2N\varepsilon^2}$$

# The intuition behind UCB

The UCB index captures both: the estimated expected reward value
& the uncertainty about this estimation

$$\hat{r}_i(t) + C\sqrt{\frac{2\ln t}{n_i(t)}}$$

$$P\left(\mu_i > \hat{r}_i(t) + C\sqrt{\frac{2\ln t}{n_i(t)}}\right)$$ is very small (by Hoeffding)

Choosing the highest UCB index captures both: exploitation and exploration

- high UCB index = worth to exploit (high expected reward)
- or worth to explore (high uncertainty)

# Regret bounds

**Upper bound** of the regret:

**Theorem1:** the expected regret of UCB is at most *O(lnT)*

**Lower bound** of the regret:

Let F(T) is a function of T: F(T) is a lower bound of the regret, if no algorithm can achieve better regret than F(T)

More precisely: for any algorithm A, there exists a setting S(A), such that within that setting, A cannot achieve better (i.e., lower) regret than F(T)

The regret lower bound of the multi-armed bandits is $\Theta(\ln T)$

**Implication:** the regret bound of UCB is *asymptotically* optimal

# Proof Sketch

Simplified proof: 2 arms, C = 1 (i.e., rewards are between 0 and 1)

W.l.o.g. assume that Arm 1 is better than Arm 2: $\mu_1 > \mu_2$

Let $\Delta = \mu_1 - \mu_2$

**Lemma 1:** the expected number of pulls of Arm 2 is at most $8\frac{\ln T}{\Delta^2} + (1 + \frac{\pi^2}{3})$

**Theorem 2:** the expected regret of UCB is at most $\mathcal{O}(\ln T)$ – *proof: use Lemma 1*

# Proof Sketch of Lemma 1

At each t, when we pull Arm 2 instead of Arm 1? (i.e., we pull the sub-optimal arm?)

Answer: when UCB index of Arm 2 > UCB index of Arm 1

Let $\mathrm{UCB}_i = \hat{r}_i(t) + \sqrt{\frac{2 \ln t}{N_i(t)}}$

We now provide upper bound for $P(\mathrm{UCB}_2 > \mathrm{UCB}_1)$

# Proof Sketch of Lemma 1

Claim: $$P(\mathrm{UCB}_2 > \mathrm{UCB}_1) \leq P(\mathrm{UCB}_1 < \mu_1) + P(\mathrm{UCB}_2 - 2\sqrt{\frac{2\ln t}{N_2(t)}} > \mu_2)$$

$$+ P(2\sqrt{\frac{2\ln t}{N_2(t)}} > \Delta)$$

Proof: if UCB2 > UCB1 happens, then at least one of these also holds:

$$\mathrm{UCB}_1 < \mu_1$$

$$\mathrm{UCB}_2 - 2\sqrt{\frac{2\ln t}{N_2(t)}} > \mu_2$$

$$2\sqrt{\frac{2\ln t}{N_2(t)}} > \Delta$$

# Proof Sketch of Lemma 1

Claim: $P(\mathrm{UCB}_1 < \mu_1) \leq t^{-4}$

Proof: use Hoeffding's inequality: $P(\hat{x}_N > \mu + \varepsilon) \leq e^{-2N\varepsilon^2}$

$$\hat{r}_1(t) + \sqrt{\frac{2\ln t}{N_1(t)}} < \mu_1$$

$$\hat{r}_1(t) < \mu_1 - \sqrt{\frac{2\ln t}{N_1(t)}}$$

$$P(\hat{r}_1(t) < \mu_1 - \sqrt{\frac{2\ln t}{N_1(t)}}) \leq t^{-4} \quad \text{(from Hoeffding)}$$

# Proof Sketch of Lemma 1

Similarly: $P(\mathrm{UCB}_2 - 2\sqrt{\frac{2\ln t}{N_2(t)}} > \mu_2) \leq t^{-4}$

Finally, we investigate the case of $2\sqrt{\frac{2\ln t}{N_2(t)}} > \Delta$

We have that if $N_2(t) > 8\frac{\ln t}{\Delta^2}$ , this cannot happen because:

$$2\sqrt{\frac{\ln t}{N_2(t)}} < 2\sqrt{\frac{\ln t}{8\frac{2\ln t}{\Delta^2}}} = \Delta$$

# Proof Sketch of Lemma 1

Why this is good for us?

Expected number of pulls of Arm 2: $\sum_{t=1}^{T} P(\mathrm{UCB}_2 > \mathrm{UCB}_1)$

If number of pulls of Arm 2 is large enough, say $N_2(t) > 8\frac{\ln t}{\Delta^2}$, we have that UCB2 > UCB1 is very unlikely (~ 2t^{-4})

Carefully summing up -> we get the upper bound of $8\frac{\ln T}{\Delta^2} + (1 + \frac{\pi^2}{3})$

Note: well known fact (using Riemann's Zeta function):

$$\sum_{t=1}^{T} 2t^{-4} < 1 + \frac{\pi^2}{3}$$

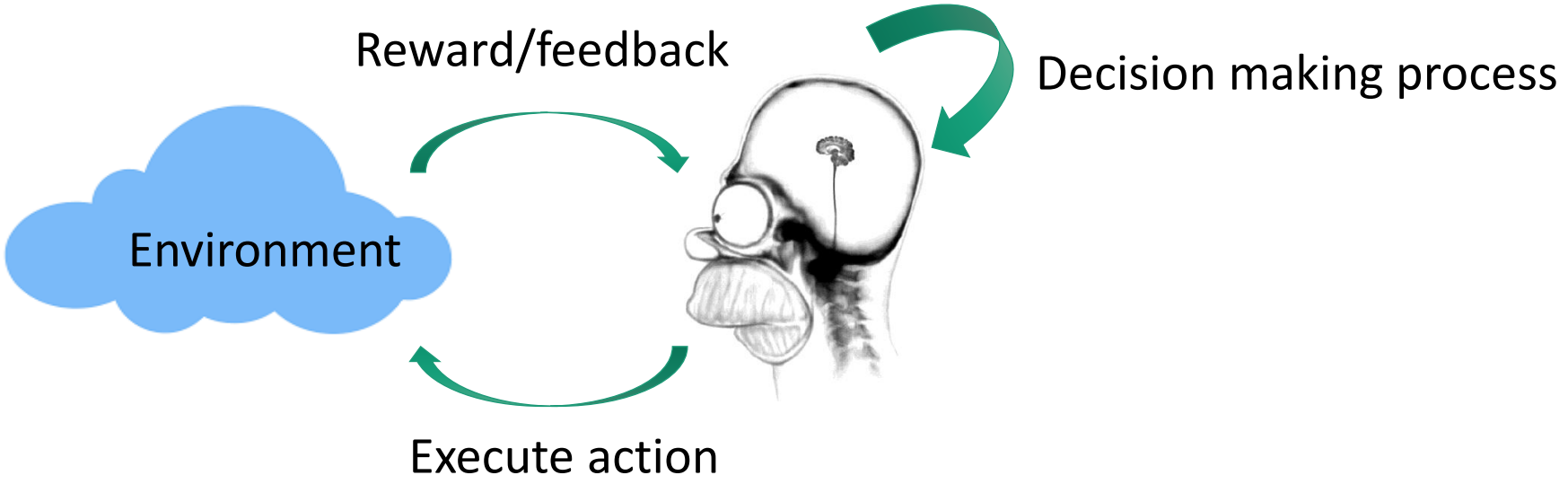
Q.E.D.

# Summary: (stochastic) multi-armed bandits

- Very simple model for sequential decision making under uncertainty
- Captures the exploration vs. exploitation dilemma
- Efficient algorithms: UCB (and many more)
- Many application domains: internet search (Bing), recommender systems, online advertisement (keyword bidding), hyperparameter optimisation in ML, discrete optimisation, ...

# Adversarial Bandits

# Motivation 1: learning in dynamic environments




Environment:

- Stochastic: governed by some underlying stationary stochastic process
- Stochastic bandits: the bandit models we've learnt so far

What about more dynamic/hostile environments?

# Motivation 2: learning against “smart” opponents

Learning against nature:

- Oblivious -> doesn’t care about your actions
- Typically stochastic environment

Learning against opponents who can also learn:

- Same game is (repeatedly) played multiple times
- If the opponent follows some underlying policy (e.g., she always chooses B if we choose A), then we can aim to learn this from the previous observations

# Non-stochastic setting

Non-adversarial:

- Changes in the environment/nature: ranging from slow and smooth to rapid and hectic
- E.g., control systems, real-world (natural) environments

Adversarial:

- Malicious: adaptive, non-adaptive
- E.g., repeated games

# Why stochastic bandit algorithms don’t work?

Iterated Rock-Paper-Scissor

- Opponent: chooses Rock in the first 100 rounds, then plays Paper in the next 200s, then Scissor in the following 300s, etc…
- We don’t have this information
- Stochastic bandit algorithms (e.g., UCB) won’t be able to cope with these changes (Why?)

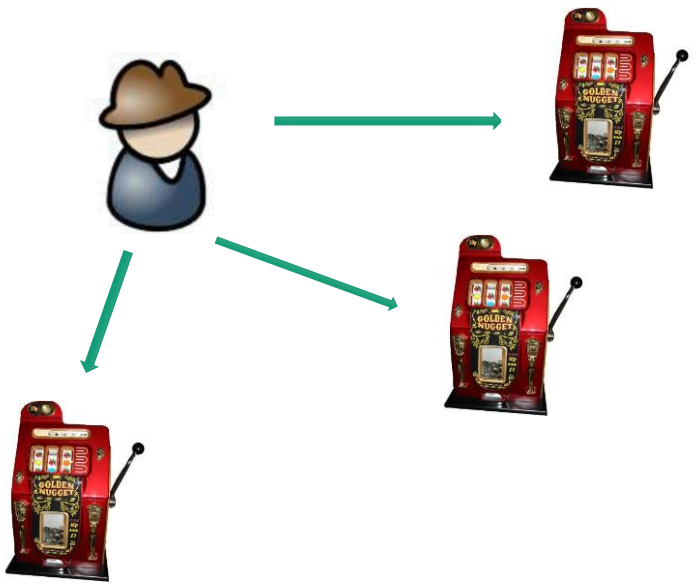
# Why stochastic bandit algorithms don’t work?

- Stochastic bandit algorithm with low regret bound = after a certain point, we rarely pull a sub-optimal arm
- This is learnt based on previous feedback
- If the environment changes, we can only slowly adapt to the changes, or cannot even detect it

Workaround solutions: time window, forgetting factor, etc…

- Only work for restricted settings

# Adversarial bandits



There are K arms

Adversarial opponent

At each time step (round):

- The adversary assigns reward values to the arms
- We choose 1 arm to pull: receive a reward chosen by the opponent

Important: the opponent has to assign the rewards of that round before we choose!!!

The opponent can be oblivious: she doesn't care about our decision making process

The opponent can be adaptive: she can see our previous choices + rewards, and adapt the assigned values based on this

# Example

Adversarial opponent has to assign a total of 5 between 2 arms

Previous rounds:

| | t=1 | t=2 | t=3 | t=4 | t=5 |
|---|---|---|---|---|---|
| Arm 1 | 3 | 4 | **1.5** | 0 | **4** |
| Arm 2 | **2** | **1** | 3.5 | **5** | 1 |

Oblivious opponent:

- Assigns Arm1 = 5, Arm2 = 3 (no particular reason)
- In general: the opponent assigns the values to all time steps before the game starts

Adaptive opponent:

- Arm2 was played more often: Arm1 = 5, Arm2 = 0

# Difficulties in adversarial bandits

Optimal solution: have full prior knowledge about the assignment per each time step of the opponent

- Very difficult to learn, even against oblivious opponents
- One reason: historical data does not give much help in predicting the future (no statistical correlations to learn)
- Consequence: the regret against the optimal solution can be arbitrarily bad
- We need different regret definitions

# Internal vs. external regret

External regret: we compare our algorithm against the best fixed arm policy in hindsight

- Fixed policy: we repeatedly pull the same arm
- Best fixed policy: not efficient as the optimal solution, but still a non-trivial (& intelligent) policy

Internal regret: we compare our algorithm against another one which only differ from ours in the following way:

- We choose two particular arms, say a1 and a2
- We replace every pull of a1 with a2 in the new algorithm
- Internal regret = comparison against the best possible replacement in hindsight

# Internal vs. external regret

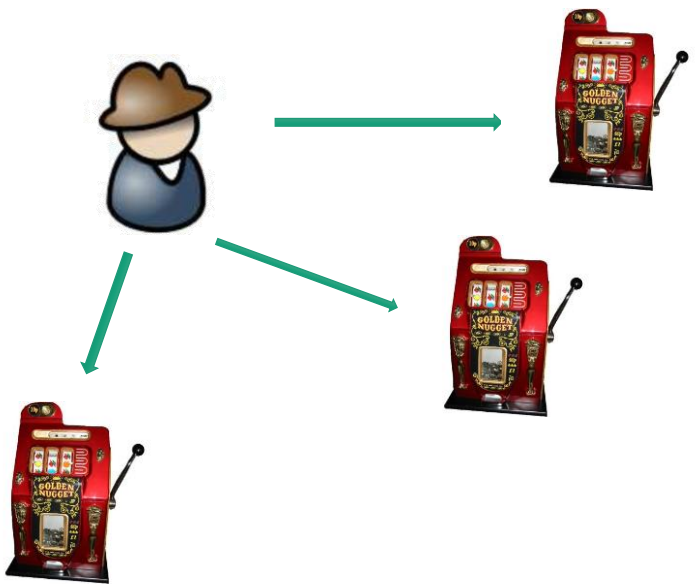
We focus on **external regret**

- Original regret notation (Hannan 1957), whereas internal regret was introduced by Foster & Vohra (1998)

- Blum & Mansour (2007) showed that the two are interchangeable:

  - Any algorithm with efficient internal regret can be modified to have low external regret, and *vice versa*

# Formal model of adversarial bandits



K arms = *{1,2,…,K}*

At each time step *t*:

Opponent chooses a vector:

$$x(t) = \{x_1(t), x_2(t), \ldots, x_K(t)\} \in [0,1]^K$$

We choose an arm:

$$i_t \in \{1, 2, \ldots, K\}$$

Objective: find a sequence of pulls $\mathcal{A} = \{i_1, i_2, \ldots, i_T\}$

$$\mathcal{A}^* = \arg\min_{\mathcal{A}} \left\{ \max_k \sum_{t=1}^{T} x_k(t) - \mathbb{E}\left[\sum_{t=1}^{T} x_{i_t}(t)\right] \right\}$$

Best fixed policy on hindsight

A can be randomized

# The Exp3 algorithm

**Parameters:** Real $\gamma \in (0, 1]$
**Initialization:** $w_i(1) = 1$ for $i = 1, \ldots, K$.

**For each** $t = 1, 2, \ldots$

1. Set

$$p_i(t) = (1 - \gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K} \qquad i = 1, \ldots, K.$$

2. Draw $i_t$ randomly accordingly to the probabilities $p_1(t), \ldots, p_K(t)$.

3. Receive reward $x_{i_t}(t) \in [0, 1]$.

4. For $j = 1, \ldots, K$ set

$$\hat{x}_j(t) = \begin{cases} x_j(t)/p_j(t) & \text{if } j = i_t \\ 0 & \text{otherwise,} \end{cases}$$

$$w_j(t+1) = w_j(t) \exp\left(\gamma \hat{x}_j(t)/K\right) .$$

# The Exp3 algorithm explained

Updating the weights

$$\hat{x}_j(t) = \begin{cases} x_j(t)/p_j(t) & \text{if } j = i_t \\ 0 & \text{otherwise,} \end{cases}$$
$$w_j(t+1) = w_j(t)\, \exp\left(\gamma \hat{x}_j(t)/K\right) .$$

- Weight value remains the same if arm not pulled (exponential part = 1)
- Exponential growth significantly increases weight of good arms
  - Arms which are better in the best has higher weights
  - Recall that we want to learn the best arm in hindsight

# The Exp3 algorithm explained

Pulling probability of arm *i* at time step *t* :

$$p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K}$$

With probability $(1-\gamma)$, it prefers arms with higher weights (exploit)

With probability $\gamma$ , chooses to uniformly randomly explore

Exp3 randomly pulls an arm, receives the rewards, then updates the weights

# Regret analysis of Exp3

Theorem: the (external) regret of Exp3 is at most: $O(\sqrt{KT \log K})$

Theorem: the (external) regret for any bandit algorithm is at least $\Theta(\sqrt{KT})$

# The Follow the Perturbed Leader (FPL) algorithm

Parameters: $\eta$

Initialisation: $\forall i : R_i(1) = 0$

For each *t = 1,2,…T*:

- For each arm generate a random noise from an exponential distribution:

$$\forall i : Z_i(t) \sim \mathrm{Exp}(\eta)$$

- Pull arm *I(t)*: $I(t) = \arg\max_i \{R_i(t) + Z_i(t)\}$
  - Add noise to each arm and pull the one with the highest value
- Update value: $R_{I(t)}(t+1) = R_{I(t)}(t) + x_{I(t)}(t)$
  - The rest remains the same

# The FPL algorithm explained

We follow the arm that we think has the best performance so far

- We add the exponential noise to it to provide exploration
- Hence the name

Intuition behind this: again, we want to learn the best arm in hindsight

Properties:

- Much simpler than Exp3, but has worse performance guarantees
- But improved versions have efficient regret bounds too!

# Exp3 vs. FPL

## Exp3

- Maintains weights for each arm to calculate pulling probability
- Has efficient theoretical guarantees
- Might be computationally expensive (calculating the exponential terms)

## FPL

- Doesn’t need to know the pulling probability per arm
- The standard FPL does not have good theoretical guarantees
- Computationally quite efficient

# Monte-Carlo Tree Search + UTC

# Simulation based search in RL

- World simulator: we can simulate the underlying MDP
- That is, we can simulate episodes of experience from now with the model starting from any current state *s*
- We can apply model-free RL to simulated episodes
    - Monte-Carlo learning -> Monte-Carlo search
    - SARSA -> TD search

# Monte-Carlo search

- World simulator: we can simulate the underlying MDP
- That is, we can simulate episodes of experience from now with the model starting from any current state $s_t$

- Evaluate action value by average return
- Select current action with the highest Q-value: $\hat{Q}(s_t, a) = \frac{1}{K} \sum_i G_{i,t}$
- This is actually 1-step lookahead calculation $a_t = \arg\max_a \hat{Q}(s_t, a)$

# Forward search with expectimax tree

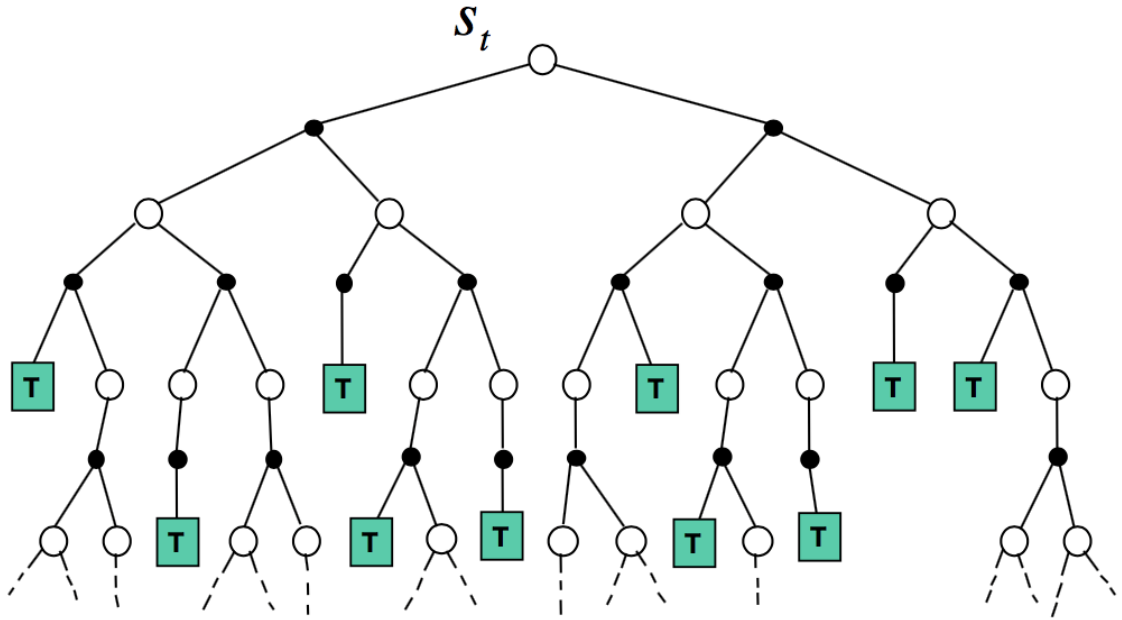
Question: can we do better than 1-step lookahead?

# Forward search with expectimax tree

Question: can we do better than 1-step lookahead?

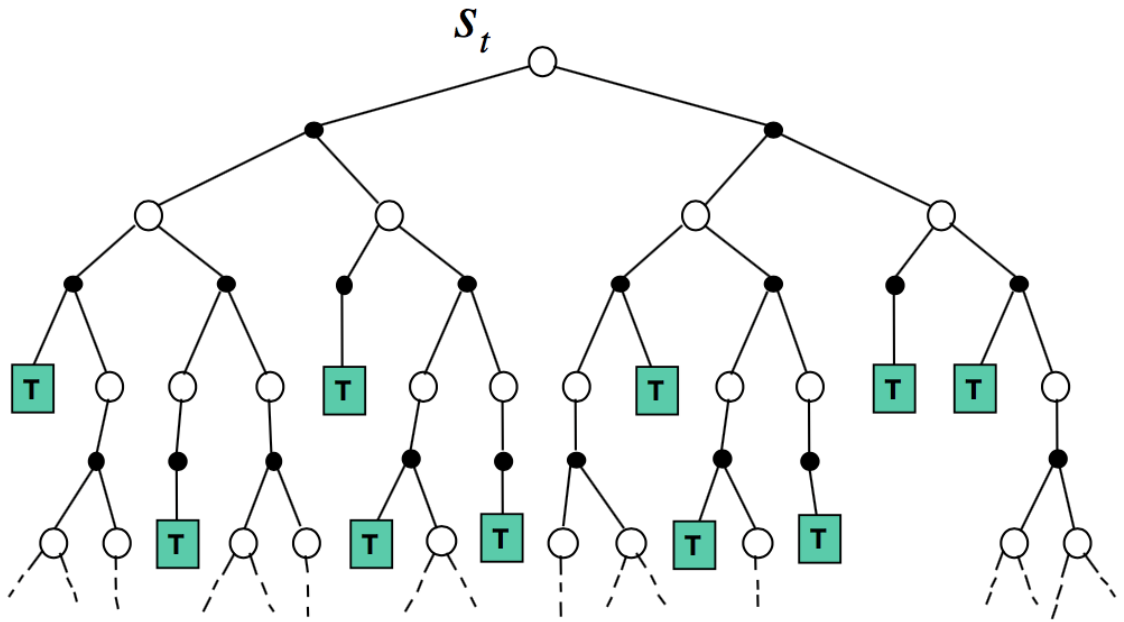
Forward search: select the best action by full lookahead

- Fully build a search tree with the current state *s_t* at the root
- Using a model of the MDP to look ahead
- No need to solve whole MDP, just sub-MDP starting from now



# Forward search with expectimax tree

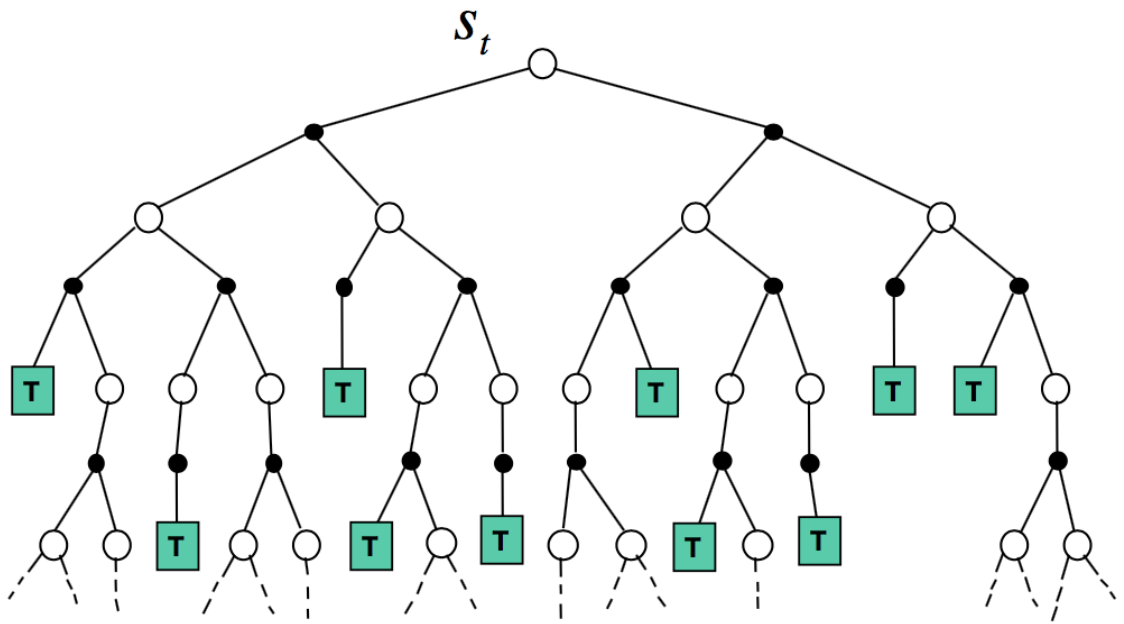
Main issue: size of sub-tree can be very large: $(|\mathcal{S}||\mathcal{A}|)^H$



# Monte-Carlo tree search (MCTS)

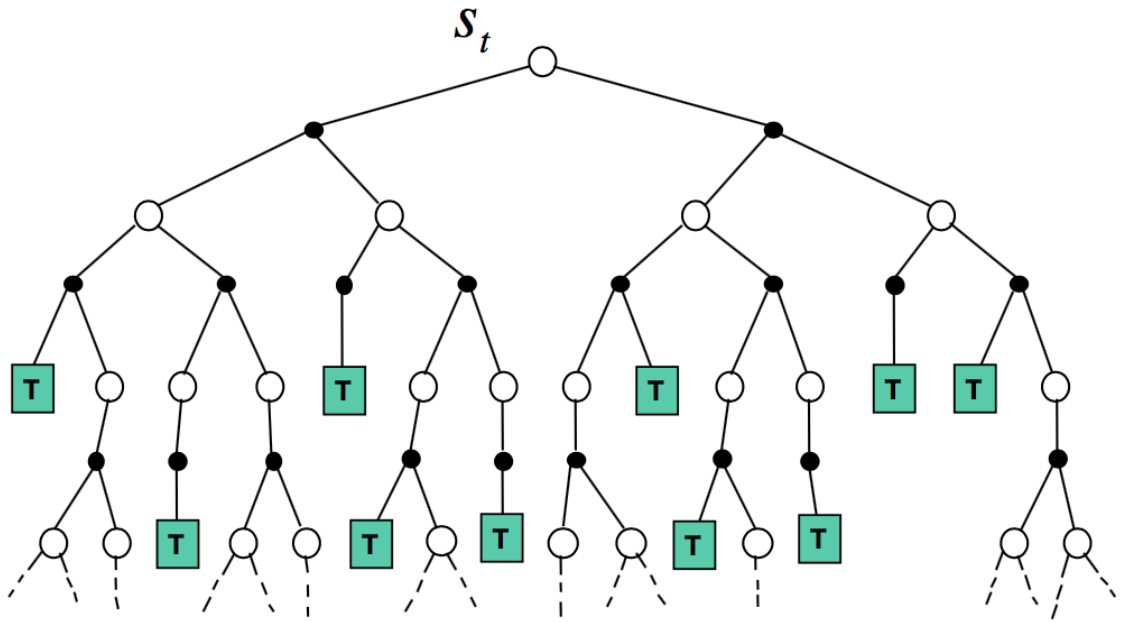
- Similarly to expectimax trees, it also builds a tree rooted at current state s_t
- But it **samples actions and next states**, instead of fully build the tree
- Performs K simulation episodes starting from the root state
- After K simulations, select current (real) action with maximum value in search tree:

$$a_t = \arg\max_a \hat{Q}(s_t, a)$$



# Upper confidence bound in trees (UCT)

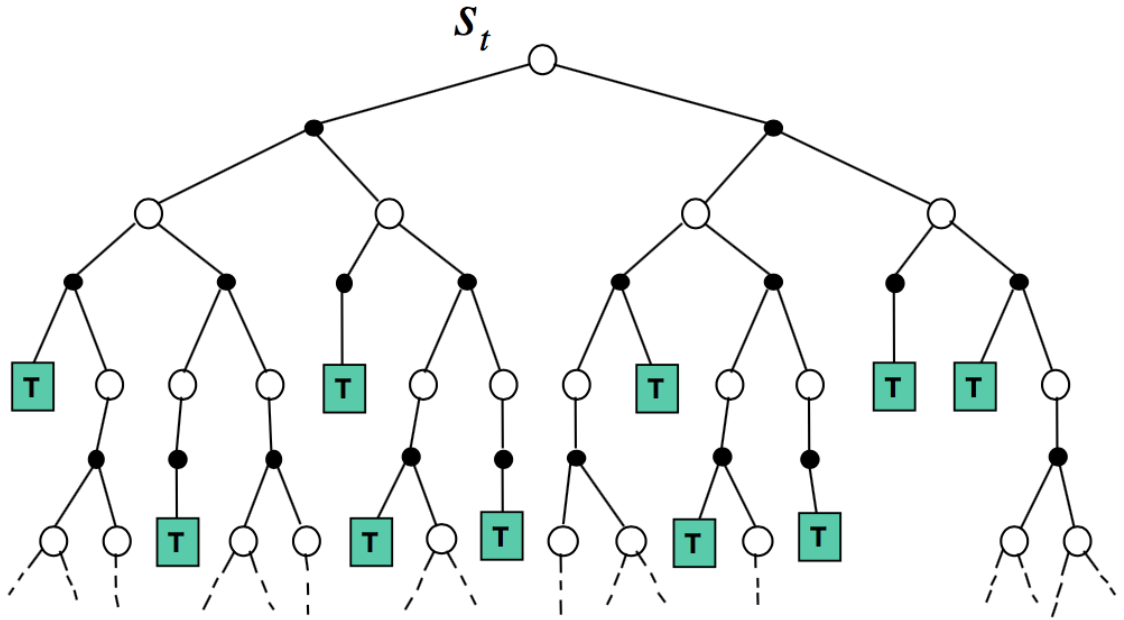
In MCTS, we do the simulation part with uniformly randomly choosing actions
Question: can we do more efficiently than that?



# Upper confidence bound in trees (UCT)

In MCTS, we do the simulation part with uniformly randomly choosing actions
Question: can we do more efficiently than that?
UCT:

- Borrow idea from bandit literature and treat each node where can select actions as a multi-armed bandit (MAB) problem
- Maintain an upper confidence bound (UCB) over reward of each arm



# Application of UCT: