

# Multi-Armed Bandits: Exploration versus Exploitation

Ashwin Rao

ICME, Stanford University

April 7, 2020

# Exploration versus Exploitation

- Many situations in business (& life!) present dilemma on choices
- **Exploitation:** Pick choices that *seem* best based on past outcomes
- **Exploration:** Pick choices not yet tried out (or not tried enough)
- Exploitation has notions of “being greedy” and being “short-sighted”
- Too much Exploitation  $\Rightarrow$  Regret of missing unexplored “gems”
- Exploration has notions of “gaining info” and being “long-sighted”
- Too much Exploration  $\Rightarrow$  Regret of wasting time on “duds”
- How to balance Exploration and Exploitation so we combine *information-gains* and *greedy-gains* in the most optimal manner
- Can we set up this problem in a mathematically disciplined manner?

# Examples

- Restaurant Selection
  - **Exploitation:** Go to your favorite restaurant
  - **Exploration:** Try a new restaurant
- Online Banner Advertisement
  - **Exploitation:** Show the most successful advertisement
  - **Exploration:** Show a new advertisement
- Oil Drilling
  - **Exploitation:** Drill at the best known location
  - **Exploration:** Drill at a new location
- Learning to play a game
  - **Exploitation:** Play the move you believe is best
  - **Exploration:** Play an experimental move

# The Multi-Armed Bandit (MAB) Problem

- Multi-Armed Bandit is spoof name for “Many Single-Armed Bandits”
- A Multi-Armed bandit problem is a 2-tuple  $(\mathcal{A}, \mathcal{R})$
- $\mathcal{A}$  is a known set of  $m$  actions (known as “arms”)
- $\mathcal{R}^a(r) = \mathbb{P}[r|a]$  is an **unknown** probability distribution over rewards
- At each step  $t$ , the AI agent (algorithm) selects an action  $a_t \in \mathcal{A}$
- Then the environment generates a reward  $r_t \sim \mathcal{R}^{a_t}$
- The AI agent’s goal is to maximize the **Cumulative Reward**:

$$\sum_{t=1}^T r_t$$

- Can we design a strategy that does well (in Expectation) for any  $T$ ?
- Note that any selection strategy risks wasting time on “duds” while exploring and also risks missing untapped “gems” while exploiting

# Is the MAB problem a Markov Decision Process (MDP)?

- Note that the environment doesn't have a notion of *State*
- Upon pulling an arm, the arm just samples from its distribution
- However, the agent might maintain a statistic of history as its *State*
- To enable the agent to make the arm-selection (action) decision
- The action is then a (*Policy*) function of the agent's *State*
- So, agent's arm-selection strategy is basically this *Policy*
- Note that many MAB algorithms don't take this formal MDP view
- Instead, they rely on heuristic methods that don't aim to *optimize*
- They simply strive for "good" Cumulative Rewards (in Expectation)
- Note that even in a simple heuristic algorithm,  $a_t$  is a random variable simply because it is a function of past (random) rewards

# Regret

- The *Action Value*  $Q(a)$  is the (unknown) mean reward of action  $a$

$$Q(a) = \mathbb{E}[r|a]$$

- The *Optimal Value*  $V^*$  is defined as:

$$V^* = Q(a^*) = \max_{a \in \mathcal{A}} Q(s)$$

- The *Regret*  $l_t$  is the opportunity loss on a single step  $t$

$$l_t = \mathbb{E}[V^* - Q(a_t)]$$

- The *Total Regret*  $L_T$  is the total opportunity loss

$$L_T = \sum_{t=1}^T \mathbb{E}[V^* - Q(a_t)]$$

- Maximizing *Cumulative Reward* is same as Minimizing *Total Regret*

# Counting Regret

- Let  $N_t(a)$  be the (random) number of selections of  $a$  across  $t$  steps
- Define  $Count_t$  of  $a$  (for given action-selection strategy) as  $\mathbb{E}[N_t(a)]$
- Define  $Gap$   $\Delta_a$  of  $a$  as the value difference between  $a$  and optimal  $a^*$

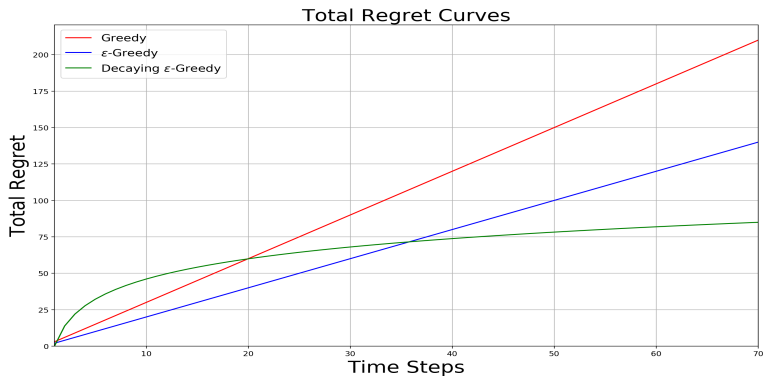
$$\Delta_a = V^* - Q(a)$$

- Total Regret is sum-product (over actions) of  $Gaps$  and  $Counts_T$

$$\begin{aligned} L_T &= \sum_{t=1}^T \mathbb{E}[V^* - Q(a_t)] \\ &= \sum_{a \in \mathcal{A}} \mathbb{E}[N_T(a)](V^* - Q(a)) \\ &= \sum_{a \in \mathcal{A}} \mathbb{E}[N_T(a)] \Delta_a \end{aligned}$$

- A good algorithm ensures small  $Counts$  for large  $Gaps$
- Little problem though: *We don't know the Gaps!*

# Linear or Sublinear Total Regret



- If an algorithm *never* explores, it will have linear total regret
- If an algorithm *forever* explores, it will have linear total regret
- Is it possible to achieve sublinear total regret?



# Greedy Algorithm

- We consider algorithms that estimate  $\hat{Q}_t(a) \approx Q(a)$
- Estimate the value of each action by rewards-averaging

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{s=1}^t r_s \cdot \mathbb{1}_{a_s=a}$$

- The *Greedy* algorithm selects the action with highest estimated value

$$a_t = \arg \max_{a \in \mathcal{A}} \hat{Q}_t(a)$$

- Greedy algorithm can lock onto a suboptimal action forever
- Hence, Greedy algorithm has linear total regret

# $\epsilon$ -Greedy Algorithm

- The  $\epsilon$ -Greedy algorithm continues to explore forever
- At each time-step  $t$ :
  - With probability  $1 - \epsilon$ , select  $a_t = \arg \max_{a \in \mathcal{A}} \hat{Q}_t(a)$
  - With probability  $\epsilon$ , select a random action (uniformly) in  $\mathcal{A}$
- Constant  $\epsilon$  ensures a minimum regret proportional to mean gap

$$I_t \geq \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \Delta_a$$

- Hence,  $\epsilon$ -Greedy algorithm has linear total regret

# Optimistic Initialization

- Simple and practical idea: Initialize  $\hat{Q}_0(a)$  to a high value for all  $a \in \mathcal{A}$
- Update action value by incremental-averaging
- Starting with  $N_0(a) \geq 0$  for all  $a \in \mathcal{A}$ ,

$$\hat{Q}_t(a_t) = \hat{Q}_{t-1}(a_t) + \frac{1}{N_t(a)}(r_t - \hat{Q}_{t-1}(a_t))$$

$$\hat{Q}_t(a) = \hat{Q}_{t-1}(a) \text{ for all } a \neq a_t$$

- Encourages systematic exploration early on
- One can also start with a high value for  $N_0(a)$
- But can still lock onto suboptimal action
- Hence, Greedy + optimistic initialization has linear total regret
- $\epsilon$ -Greedy + optimistic initialization also has linear total regret

# Decaying $\epsilon_t$ -Greedy Algorithm

- Pick a decay schedule for  $\epsilon_1, \epsilon_2, \dots$
- Consider the following schedule

$$c > 0$$

$$d = \min_{a | \Delta_a > 0} \Delta_a$$

$$\epsilon_t = \min\left(1, \frac{c|\mathcal{A}|}{d^2 t}\right\}$$

- Decaying  $\epsilon_t$ -Greedy algorithm has *logarithmic* total regret
- Unfortunately, above schedule requires advance knowledge of gaps
- Practically, implementing *some* decay schedule helps considerably
- [Educational Code](#) for decaying  $\epsilon$ -greedy with optimistic initialization

# Lower Bound

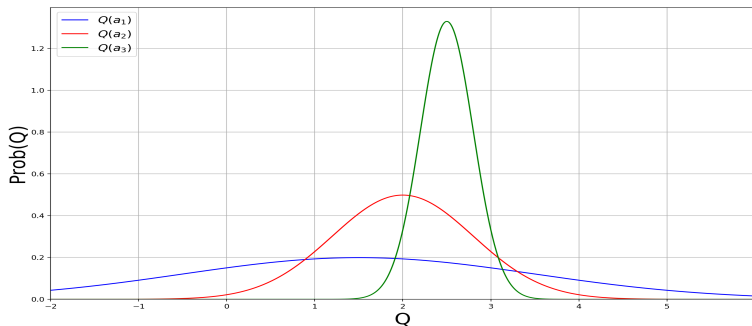
- Goal: Find an algorithm with sublinear total regret for any multi-armed bandit (without any prior knowledge of  $\mathcal{R}$ )
- The performance of any algorithm is determined by the similarity between the optimal arm and other arms
- Hard problems have similar-looking arms with different means
- Formally described by KL-Divergence  $KL(\mathcal{R}^a || \mathcal{R}^{a^*})$  and gaps  $\Delta_a$

## Theorem (Lai and Robbins)

*Asymptotic Total Regret is at least logarithmic in number of steps*

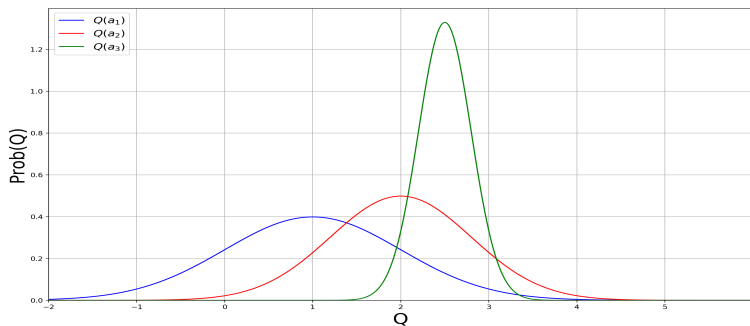
$$\lim_{T \rightarrow \infty} L_T \geq \log T \sum_{a | \Delta_a > 0} \frac{1}{\Delta_a} \geq \log T \sum_{a | \Delta_a > 0} \frac{\Delta_a}{KL(\mathcal{R}^a || \mathcal{R}^{a^*})}$$

# Optimism in the Face of Uncertainty



- Which action should we pick?
- The more uncertain we are about an action-value, the more important it is to explore that action
- It could turn out to be the best action

# Optimism in the Face of Uncertainty (continued)



- After picking *blue* action, we are less uncertain about the value
- And more likely to pick another action
- Until we home in on the best action

# Upper Confidence Bounds

- Estimate an upper confidence  $\hat{U}_t(a)$  for each action value
- Such that  $Q(a) \leq \hat{Q}_t(a) + \hat{U}_t(a)$  with high probability
- This depends on the number of times  $N_t(a)$  that  $a$  has been selected
  - Small  $N_t(a) \Rightarrow$  Large  $\hat{U}_t(a)$  (estimated value is uncertain)
  - Large  $N_t(a) \Rightarrow$  Small  $\hat{U}_t(a)$  (estimated value is accurate)
- Select action maximizing Upper Confidence Bound (UCB)

$$a_t = \arg \max_{a \in \mathcal{A}} \{ \hat{Q}_t(a) + \hat{U}_t(a) \}$$



# Hoeffding's Inequality

## Theorem (Hoeffding's Inequality)

Let  $X_1, \dots, X_t$  be i.i.d. random variables in  $[0, 1]$ , and let

$$\bar{X}_t = \frac{1}{t} \sum_{s=1}^t X_s$$

be the sample mean. Then,

$$\mathbb{P}[\mathbb{E}[X] > \bar{X}_t + u] \leq e^{-2tu^2}$$

- Apply Hoeffding's Inequality to rewards of  $[0, 1]$ -support bandits
- Conditioned on selecting action  $a$

$$\mathbb{P}[Q(a) > \hat{Q}_t(a) + \hat{U}_t(a)] \leq e^{-2N_t(a)\hat{U}_t(a)^2}$$

# Calculating Upper Confidence Bounds

- Pick a small probability  $p$  that  $Q(a)$  exceeds UCB  $\{\hat{Q}_t(a) + \hat{U}_t(a)\}$
- Now solve for  $\hat{U}_t(a)$

$$e^{-2N_t(a)\hat{U}_t(a)^2} = p$$

$$\Rightarrow \hat{U}_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

- Reduce  $p$  as we observe more rewards, eg:  $p = t^{-\alpha}$  (for fixed  $\alpha > 0$ )
- This ensures we select optimal action as  $t \rightarrow \infty$

$$\hat{U}_t(a) = \sqrt{\frac{\alpha \log t}{2N_t(a)}}$$

Yields UCB1 algorithm for arbitrary-distribution arms bounded in  $[0, 1]$

$$a_t = \arg \max_{a \in \mathcal{A}} \left\{ \hat{Q}_t(a) + \sqrt{\frac{\alpha \log t}{2N_t(a)}} \right\}$$

## Theorem

*The UCB1 Algorithm achieves logarithmic total regret*

$$L_T \leq \sum_{a | \Delta_a > 0} \frac{4\alpha \cdot \log T}{\Delta_a} + \frac{2\alpha \cdot \Delta_a}{\alpha - 1}$$

[Educational Code](#) for UCB1 algorithm

- So far we have made no assumptions about the rewards distribution  $\mathcal{R}$  (except bounds on rewards)
- *Bayesian Bandits* exploit prior knowledge of rewards distribution  $\mathbb{P}[\mathcal{R}]$
- They compute posterior distribution of rewards  $\mathbb{P}[\mathcal{R}|h_t]$  where  $h_t = a_1, r_1, \dots, a_{t-1}, r_{t-1}$  is the history
- Use posterior to guide exploration
  - Upper Confidence Bounds (Bayesian UCB)
  - Probability Matching (Thompson sampling)
- Better performance if prior knowledge of  $\mathcal{R}$  is accurate

# Bayesian UCB Example: Independent Gaussians

- Assume reward distribution is Gaussian,  $\mathcal{R}^a(r) = \mathcal{N}(r; \mu_a, \sigma_a^2)$
- Compute Gaussian posterior over  $\mu_a, \sigma_a^2$  (Bayes update details [here](#))

$$\mathbb{P}[\mu_a, \sigma_a^2 | h_t] \propto \mathbb{P}[\mu_a, \sigma_a^2] \cdot \prod_{t|a_t=a} \mathcal{N}(r_t; \mu_a, \sigma_a^2)$$

- Pick action that maximizes Expectation of  $c$  std-devs above mean

$$a_t = \arg \max_{a \in \mathcal{A}} \mathbb{E}[\mu_a + \frac{c\sigma_a}{\sqrt{N_t(a)}}]$$

- *Probability Matching* selects action  $a$  according to probability that  $a$  is the optimal action

$$\pi(a|h_t) = \mathbb{P}[Q(a) > Q(a'), \forall a' \neq a | h_t]$$

- Probability matching is optimistic in the face of uncertainty
- Because uncertain actions have higher probability of being max
- Can be difficult to compute analytically from posterior

# Thompson Sampling

- *Thompson Sampling* implements probability matching

$$\pi(a|h_t) = \mathbb{P}[Q(a) > Q(a'), \forall a' \neq a | h_t]$$

$$= \mathbb{E}_{\mathcal{R}|h_t}[\mathbb{1}_{a=\arg \max_{a \in \mathcal{A}} Q(a)}]$$

- Use Bayes law to compute posterior distribution  $\mathbb{P}[\mathcal{R}|h_t]$
- *Sample* a reward distribution  $\mathcal{R}$  from posterior
- Compute Action-Value function  $Q(a) = \mathbb{E}_{\mathcal{R}^a}[r]$
- Select action maximizing value of sample

$$a_t = \arg \max_{a \in \mathcal{A}} Q(a)$$

- Thompson Sampling achieves Lai-Robbins lower bound!
- [Educational Code](#) for Thompson Sampling for Gaussian Distributions
- [Educational Code](#) for Thompson Sampling for Bernoulli Distributions

# Gradient Bandit Algorithms

- Gradient Bandit Algorithms are based on Stochastic Gradient Ascent
- We optimize *Score* parameters  $s_a$  for  $a \in \mathcal{A} = \{a_1, \dots, a_m\}$
- Objective function to be maximized is the Expected Reward

$$J(s_{a_1}, \dots, s_{a_m}) = \sum_{a \in \mathcal{A}} \pi(a) \cdot \mathbb{E}[r|a]$$

- $\pi(\cdot)$  is probabilities of taking actions (based on a stochastic policy)
- The stochastic policy governing  $\pi(\cdot)$  is a function of the *Scores*:

$$\pi(a) = \frac{e^{s_a}}{\sum_{b \in \mathcal{A}} e^{s_b}}$$

- *Scores* represent the relative value of actions based on seen rewards
- Note:  $\pi$  has a Boltzmann distribution (Softmax-function of *Scores*)
- We move the *Score* parameters  $s_a$  (hence, action probabilities  $\pi(a)$ ) such that we ascend along the direction of gradient of objective  $J(\cdot)$



# Gradient of Expected Reward

- To construct Gradient of  $J(\cdot)$ , we calculate  $\frac{\partial J}{\partial s_a}$  for all  $a \in \mathcal{A}$

$$\begin{aligned}\frac{\partial J}{\partial s_a} &= \frac{\partial}{\partial s_a} \left( \sum_{a' \in \mathcal{A}} \pi(a') \cdot \mathbb{E}[r|a'] \right) = \sum_{a' \in \mathcal{A}} \mathbb{E}[r|a'] \cdot \frac{\partial \pi(a')}{\partial s_a} \\ &= \sum_{a' \in \mathcal{A}} \pi(a') \cdot \mathbb{E}[r|a'] \cdot \frac{\partial \log \pi(a')}{\partial s_a} = \mathbb{E}_{a' \sim \pi, r \sim \mathcal{R}^{a'}} \left[ r \cdot \frac{\partial \log \pi(a')}{\partial s_a} \right]\end{aligned}$$

- We know from standard softmax-function calculus that:

$$\frac{\partial \log \pi(a')}{\partial s_a} = \frac{\partial}{\partial s_a} \left( \log \frac{e^{s_{a'}}}{\sum_{b \in \mathcal{A}} e^{s_b}} \right) = \mathbb{1}_{a=a'} - \pi(a)$$

- Therefore  $\frac{\partial J}{\partial s_a}$  can be re-written as:

$$= \mathbb{E}_{a' \sim \pi, r \sim \mathcal{R}^{a'}} [r \cdot (\mathbb{1}_{a=a'} - \pi(a))]$$

- At each step  $t$ , we approximate the gradient with  $(a_t, r_t)$  sample as:

$$r_t \cdot (\mathbb{1}_{a=a_t} - \pi_t(a)) \text{ for all } a \in \mathcal{A}$$

# Score updates with Stochastic Gradient Ascent

- $\pi_t(a)$  is the probability of  $a$  at step  $t$  derived from score  $s_t(a)$  at step  $t$
- Reduce variance of estimate with baseline  $B$  that's independent of  $a$ :

$$(r_t - B) \cdot (\mathbb{1}_{a=a_t} - \pi_t(a)) \text{ for all } a \in \mathcal{A}$$

- This doesn't introduce bias in the estimate of gradient of  $J(\cdot)$  because

$$\begin{aligned}\mathbb{E}_{a' \sim \pi}[B \cdot (\mathbb{1}_{a=a'} - \pi(a))] &= \mathbb{E}_{a' \sim \pi}[B \cdot \frac{\partial \log \pi(a')}{\partial s_a}] \\ &= B \cdot \sum_{a' \in \mathcal{A}} \pi(a') \cdot \frac{\partial \log \pi(a')}{\partial s_a} = B \cdot \sum_{a' \in \mathcal{A}} \frac{\partial \pi(a')}{\partial s_a} = B \cdot \frac{\partial}{\partial s_a} \left( \sum_{a' \in \mathcal{A}} \pi(a') \right) = 0\end{aligned}$$

- We can use  $B = \bar{r}_t = \frac{1}{t} \sum_{s=1}^t r_s$  = average rewards until step  $t$
- So, the update to scores  $s_t(a)$  for all  $a \in \mathcal{A}$  is:

$$s_{t+1}(a) = s_t(a) + \alpha \cdot (r_t - \bar{r}_t) \cdot (\mathbb{1}_{a=a_t} - \pi_t(a))$$

- [Educational Code](#) for this Gradient Bandit Algorithm

- Exploration is useful because it gains information
- Can we quantify the value of information?
  - How much would a decision-maker be willing to pay to have that information, prior to making a decision?
  - Long-term reward after getting information minus immediate reward
- Information gain is higher in uncertain situations
- Therefore it makes sense to explore uncertain situations more
- If we know value of information, we can trade-off exploration and exploitation *optimally*

# Information State Space

- We have viewed bandits as *one-step* decision-making problems
- Can also view as *sequential* decision-making problems
- At each step there is an *information state*  $\tilde{s}$ 
  - $\tilde{s}$  is a statistic of the history, i.e.,  $\tilde{s}_t = f(h_t)$
  - summarizing all information accumulated so far
- Each action  $a$  causes a transition to a new information state  $\tilde{s}'$  (by adding information), with probability  $\tilde{\mathcal{P}}_{\tilde{s}, \tilde{s}'}^a$
- This defines an MDP  $\tilde{M}$  in information state space

$$\tilde{M} = (\tilde{\mathcal{S}}, \mathcal{A}, \tilde{\mathcal{P}}, \mathcal{R}, \gamma)$$

## Example: Bernoulli Bandits

- Consider a Bernoulli Bandit, such that  $\mathcal{R}^a = \mathcal{B}(\mu_a)$
- For arm  $a$ , reward=1 with probability  $\mu_a$  (=0 with probability  $1 - \mu_a$ )
- Assume we have  $m$  arms  $a_1, a_2, \dots, a_m$
- The information state is  $\tilde{s} = (\alpha_{a_1}, \beta_{a_1}, \alpha_{a_2}, \beta_{a_2}, \dots, \alpha_{a_m}, \beta_{a_m})$
- $\alpha_a$  records the pulls of arms  $a$  for which reward was 1
- $\beta_a$  records the pulls of arm  $a$  for which reward was 0
- In the long-run,  $\frac{\alpha_a}{\alpha_a + \beta_a} \rightarrow \mu_a$

# Solving Information State Space Bandits

- We now have an infinite MDP over information states
- This MDP can be solved by Reinforcement Learning
- Model-free Reinforcement learning, eg: Q-Learning (Duff, 1994)
- Or Bayesian Model-based Reinforcement Learning
  - eg: Gittins indices (Gittins, 1979)
  - This approach is known as Bayes-adaptive RL
  - Finds Bayes-optimal exploration/exploitation trade-off with respect of prior distribution

# Bayes-Adaptive Bernoulli Bandits

- Start with  $Beta(\alpha_a, \beta_a)$  prior over reward function  $\mathcal{R}^a$
- Each time  $a$  is selected, update posterior for  $\mathcal{R}^a$  as:
  - $Beta(\alpha_a + 1, \beta_a)$  if  $r = 1$
  - $Beta(\alpha_a, \beta_a + 1)$  if  $r = 0$
- This defines transition function  $\tilde{\mathcal{P}}$  for the Bayes-adaptive MDP
- $(\alpha_a, \beta_a)$  in information state provides reward model  $Beta(\alpha_a, \beta_a)$
- Each state transition corresponds to a Bayesian model update

# Gittins Indices for Bernoulli Bandits

- Bayes-adaptive MDP can be solved by Dynamic Programming
- The solution is known as the Gittins Index
- Exact solution to Bayes-adaptive MDP is typically intractable
- Guez et al. 2020 applied Simulation-based search
  - Forward search in information state space
  - Using simulations from current information state



# Summary of approaches to Bandit Algorithms

- Naive Exploration (eg:  $\epsilon$ -Greedy)
- Optimistic Initialization
- Optimism in the face of uncertainty (eg: UCB, Bayesian UCB)
- Probability Matching (eg: Thompson Sampling)
- Gradient Bandit Algorithms
- Information State Space MDP, incorporating value of information

# Contextual Bandits

- A Contextual Bandit is a 3-tuple  $(\mathcal{A}, \mathcal{S}, \mathcal{R})$
- $\mathcal{A}$  is a known set of  $m$  actions (“arms”)
- $\mathcal{S} = \mathbb{P}[s]$  is an **unknown** distribution over states (“contexts”)
- $\mathcal{R}_s^a(r) = \mathbb{P}[r|s, a]$  is an **unknown** probability distribution over rewards
- At each step  $t$ , the following sequence of events occur:
  - The environment generates a states  $s_t \sim \mathcal{S}$
  - Then the AI Agent (algorithm) selects an actions  $a_t \in \mathcal{A}$
  - Then the environment generates a reward  $r_t \in \mathcal{R}_{s_t}^{a_t}$
- The AI agent’s goal is to maximize the Cumulative Reward:

$$\sum_{t=1}^T r_t$$

- Extend Bandit Algorithms to Action-Value  $Q(s, a)$  (instead of  $Q(a)$ )