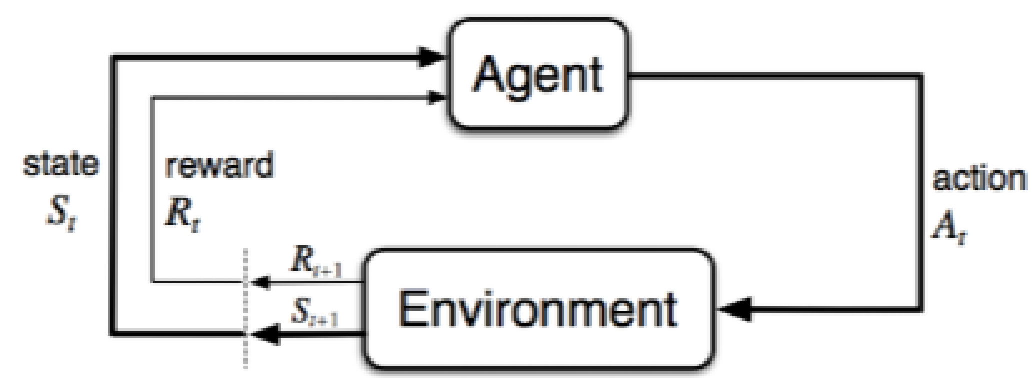
Reinforcement Learning Introduction

Jacob Chmura

1 Basic Definitions and Formulation



- **agent** is acting in an **environment**
- agent exists in one of many **states** $s \in \mathcal{S}$
- agent takes one of many **actions** $a \in \mathcal{A}$
- agents action causes a **transition** to new state $s' \in \mathcal{S}$ and environment delivers a **reward** $r \in \mathcal{R}$

Goal: learn, through the interaction between agent and environment, the optimal action to take in a given state to **maximize total rewards**

Definition 1. (Reward Hypothesis)
All goals can be described by the maximisation of expected cumulative reward

1.1 Model

The **model** is a description of the environment, via the transition map P and reward map R:

$$P^a_{s,s'} := P(s'|s,a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} P(s', r|s,a) \tag{1}$$

$$R(s,a) := \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r|s,a) \tag{2}$$

- **model-based** RL either assumes knowledge of $P^a_{s,s'}$ and $R(s,a)$ or tries to learn it explicitly
- **model-free** RL the agent learns a policy without modelling the environment dynamics

1.2 Policy

The **policy** is the agent's behaviour function:

$$\pi : \mathcal{S} \mapsto \mathcal{A} \tag{3}$$

$$\pi(a|s) = \mathbb{P}_\pi[A = a|S = s] \tag{4}$$

- Ultimately the goal is to learn a policy π that is *optimal*

- **on-policy learning** attempts to evaluate/improve the *same* policy that is being used to make decisions
- **off-policy learning** evaluate a policy while following a *different* behavioural policy (e.g. evaluate a greedy policy while following a more explorative scheme)

1.3 Value Function

The **value function** measures how rewarding a state of action is in terms of expected *future reward*

The **future reward (return)**:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{5}$$

where $\gamma \in [0, 1]$ is called the *discounting factor* which decays rewards in the future to account for uncertainty, and to simplify the math

The **state-value** is the *expected return from the given state under a policy*:

$$V_\pi(s) := \mathbb{E}_\pi[G_t | S_t = s] \tag{6}$$

The **action-value** is the *expected return from the given state taking a specific action, then following a policy*:

$$Q_\pi(s, a) := \mathbb{E}_\pi[G_t | S_t = s, A_t = s] \tag{7}$$

The **advantage** is the *difference between action-value and state value*:

$$A_\pi(s, a) := Q_\pi(s, a) - V_\pi(s) \tag{8}$$

Revisiting the goal of long term reward maximization

- Value functions define a *partial ordering* over the space of policies:

$$\pi \geq \pi' \iff V_\pi(s) \geq V_{\pi'}(s) \forall s \in \mathcal{S} \tag{9}$$

The *optimal* policy is the one achieving the *optimal value functions*, which are the value functions producing max return:

$$\pi_* = argmax_\pi V_\pi(s) = argmax_\pi Q_\pi(s, a) \tag{10}$$

$$V_*(s) = max_\pi V_\pi(s), Q_*(s, a) = max_\pi Q_\pi(s, a) \tag{11}$$

1.4 Markov Processes

The **markov property** is that the future and past are *conditionally independent given the present*:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t] \tag{12}$$

A **Markov Decision Process (MDP)** is a 5-tuple:

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle \tag{13}$$

where the state space $\mathcal{S}$ is *Markov* with respect to transition dynamics $\mathcal{P}$

- this is the standard way to formulate a RL problem

1.5 Bellman Expectation Equation

A set of recursive equation that hold for MDP's, exploited in iterative dynamic programming solutions

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s] \tag{14}$$
$$Q(s,a) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a\sim\pi} Q(S_{t+1}, a)|S_t = s, A_t = a] \tag{15}$$

Value at current position equals immediate reward plus value at next position

1.6 Bellman Equation *(can ignore)*

Further decomposition of the equations above:

$$V_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\underline{Q_\pi(s,a)} \tag{16}$$
$$Q_\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{s,s'}\underline{V_\pi(s')} \tag{17}$$
$$V_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)(\mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{s,s'} V_\pi(s')) \tag{18}$$
$$Q_\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{s,s'} \sum_{a'\in\mathcal{A}} \pi(a|s)Q_\pi(s,a) \tag{19}$$

1.7 Bellman Optimality Equation

Relationships between value functions under optimality:

$$V_*(s) = max_{a\in\mathcal{A}}(\mathcal{R}(s,a)) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{s,s'} V_*(s')) \tag{20}$$
$$Q_*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{s,s'} max_{a'\in\mathcal{A}} Q_*(s,a) \tag{21}$$

The best value at current position equals the best immediate reward plus the best value at next position

2 Value Based Approach

Idea: estimate how good states and actions are based on the expected total rewards, then follow the policy that realizes these states and actions.

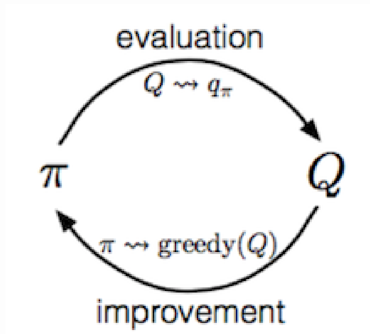
2.1 Monte-Carlo Methods

- Learn from *episodes* of experience without modelling environment dynamics
- Used *observed mean return* to approximate expected return using **complete episodes**

$$V(s) \approx \frac{\sum_{t=1}^{T} \mathbb{1}[S_t = s] G_t}{\sum_{t=1}^{T} \mathbb{1}[S_t = s]} \tag{22}$$

$$Q(s,a) \approx \frac{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a] G_t}{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a]} \tag{23}$$

$$\tag{24}$$



1. Improve the policy greedily with respect to the current value function: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$.
2. Generate a new episode with the new policy π (i.e. using algorithms like ε-greedy helps us balance between exploitation and exploration.)
3. Estimate Q using the new episode: $q_\pi(s,a) = \frac{\sum_{t=1}^{T} \left(1[S_t = s, A_t = a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \right)}{\sum_{t=1}^{T} 1[S_t = s, A_t = a]}$

Why does step (1) work? Let π be any policy, and π' be the policy induced from π be greedily taking actions:

$$\pi'(s) = argmax_{a \in \mathcal{A}} Q_\pi(s,a) \tag{25}$$

Then:

$$\begin{aligned} Q_\pi(s, \pi'(s)) &= Q_\pi(s, argmax_{a \in \mathcal{A}} Q_\pi(s,a)) && (26) \\ &= max_{a \in \mathcal{A}} Q_\pi(s,a) && (27) \\ &\geq Q_\pi(s, \pi(s)) && (28) \\ &= V_\pi(s) && (29) \end{aligned}$$

Problems

- requires full episode of experience to perform any updates
- high variance

2.2 Temporal Difference Learning

Utilize *bootstrapping* to learn from *incomplete* episodes

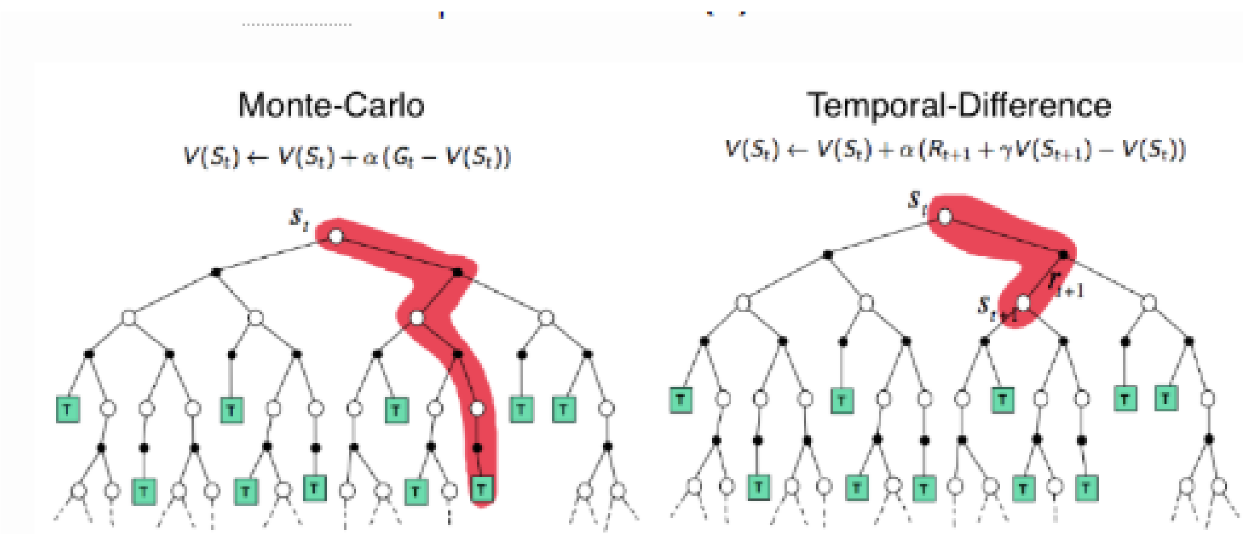
Bootstrapping involves updating targets with regard to existing estimates rather than complete returns

$$V(S_t) \leftarrow V(S_t) + \alpha(\underline{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)) \tag{30}$$

- $\underline{R_{t+1} + \gamma V(S_{t+1})}$ is called the **TD target** which is an estimate of the return: G_t

Similarly,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\underline{R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})} - Q(S_t, A_t)) \tag{31}$$



2.3 SARSA: on-policy TD-control

[1]

SARSA: On-Policy TD control

"SARSA" refers to the procedure of updaing Q-value by following a sequence of $\ldots, S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \ldots$. The idea follows the same route of GPI:

1. At time step t, we start from state S_t and pick action according to Q values, $A_t = \arg\max_{a\in\mathcal{A}} Q(S_t, a)$; ε-greedy is commonly applied.
2. With action A_t, we observe reward R_{t+1} and get into the next state S_{t+1}.
3. Then pick the next action in the same way as in step 1.: $A_{t+1} = \arg\max_{a\in\mathcal{A}} Q(S_{t+1}, a)$.
4. Update the action-value function: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$.
5. t = t+1 and repeat from step 1.

In each update of SARSA, we need to choose actions for two steps by following the current policy twice (in Step 1. & 3.).

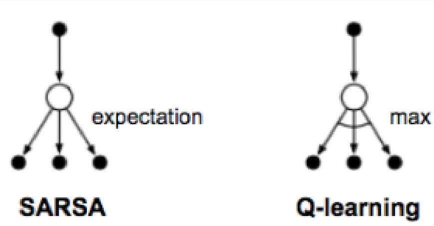
2.4 Q-learning: off-policy TD-control

Q-Learning: Off-policy TD control

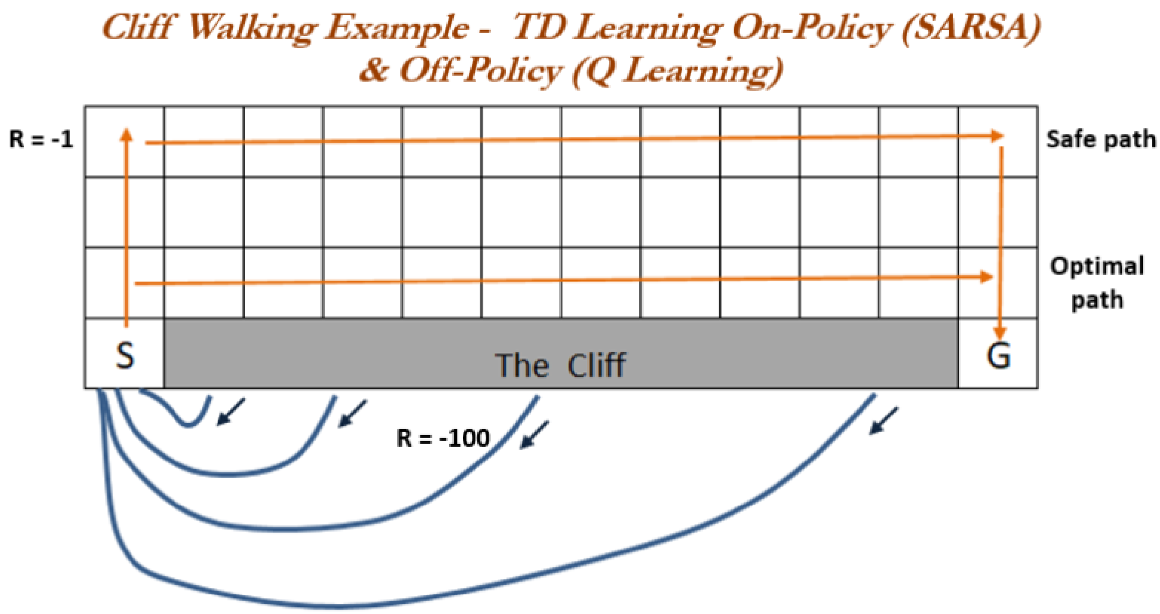
The development of Q-learning (Watkins & Dayan, 1992) is a big breakout in the early days of Reinforcement Learning.

1. At time step t, we start from state S_t and pick action according to Q values, $A_t = \arg\max_{a\in\mathcal{A}} Q(S_t, a)$; ε-greedy is commonly applied.
2. With action A_t, we observe reward R_{t+1} and get into the next state S_{t+1}.
3. Update the action-value function: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a\in\mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$.
4. t = t+1 and repeat from step 1.

The first two steps are same as in SARSA. In step 3., Q-learning does not follow the current policy to pick the second action but rather estimate Q_* out of the best Q values independently of the current policy.



[1]Typically we use a function approximator parameterized by θ like: $Q(s, a; \theta)$



- Q learning will take the optimal path, SARSA will take the safe path
- Q learning is policy-agnostic, and assumes optimality
- SARSA looks one step ahead and notices the potential to fall of the cliff, thereby reducing Q-values of neighbouring cells
- under greedy behaviour, they are equivalent

2.5 TD(λ)

Rather than bootstrapping the *one-step return* we can iterate the *n-step return*:

Let's label the estimated return following n steps as $G_t^{(n)}, n = 1, \dots, \infty$, then:

n	G_t	**Notes**
$n = 1$	$G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$	TD learning
$n = 2$	$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$	
...		
$n = n$	$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$	
...		
$n = \infty$	$G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T + \gamma^{T-t} V(S_T)$	MC estimation

The generalized n-step TD learning still has the same form for updating the value function:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$

- bias-variance tradeoff as a function of n
- alternatively: weighted average of *all* n:

$$G_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \tag{32}$$

3 Policy Gradients

Idea: learn a parameterized policy directly using optimization techniques so that expected return is maximized

==

$$\pi(a|s;\theta) \ s.t. \ \theta = argmax \mathcal{J}(\theta) \tag{33}$$

$$\mathcal{J}(\theta) = \sum_{s \in \mathcal{S}} d_{\pi_\theta}(s) V_{\pi_\theta}(s) \tag{34}$$

where d_{π_θ} is the *stationary distribution* of Markov chain under π_θ:

$$d_\pi(s) = lim_{t \to \infty} P(s_t = s | s_0, \pi) \tag{35}$$

Problem

- under the assumption that the environment is unknown, how do we differentiate $d_\pi(\cdot)$?

Theorem 3.1. *(Policy Gradient Theorem)*

$$\nabla_\theta \mathcal{J}(\theta) \propto \sum_{s \in \mathcal{S}} d_\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s) \tag{36}$$

$$= \mathbb{E}_\pi[Q^\pi(s,a) \nabla_\theta ln \pi_\theta(a|s)] \tag{37}$$

- provides equivalent optimization objective to $\mathcal{J}(\theta)$ that does not involve the derivative of the state distribution

3.1 Reinforce

REINFORCE (Monte-Carlo policy gradient) relies on an estimated return by Monte-Carlo method using episode samples to update the policy parameter θ. REINFORCE works because the expectation of the sample gradient is equal to the actual gradient:

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_\pi[Q^\pi(s,a) \nabla_\theta \ln \pi_\theta(a|s)] \\ &= \mathbb{E}_\pi[G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)] \qquad ; \text{Because } Q^\pi(S_t,A_t) = \mathbb{E}_\pi[G_t|S_t,A_t] \end{aligned}$$

Therefore we are able to measure G_t from real sample trajectories and use that to update our policy gradient. It relies on a full trajectory and that's why it is a Monte-Carlo method.

The process is pretty straightforward:

1. Initialize the policy parameter θ at random.
2. Generate one trajectory on policy π_θ: $S_1, A_1, R_2, S_2, A_2, \dots, S_T$.
3. For t=1, 2, ... , T:
    1. Estimate the the return G_t;
    2. Update policy parameters: $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$

- to reduce variance of gradient estimation while keeping bias unchanged, we typically subtract a baseline from the return G_t (for example use advantage instead of action-value)

3.2 Actor-Critic

Learn the value function in addition to the policy to reduce gradient variance.

Actor-critic methods consist of two models, which may optionally share parameters:

- **Critic** updates the value function parameters w and depending on the algorithm it could be action-value $Q_w(a|s)$ or state-value $V_w(s)$.
- **Actor** updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic.

Let's see how it works in a simple action-value actor-critic algorithm.

1. Initialize s, θ, w at random; sample $a \sim \pi_\theta(a|s)$.
2. For $t = 1 \dots T$:
    1. Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$;
    2. Then sample the next action $a' \sim \pi_\theta(a'|s')$;
    3. Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s)$;
    4. Compute the correction (TD error) for action-value at time t:
       $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
       and use it to update the parameters of action-value function:
       $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
    5. Update $a \leftarrow a'$ and $s \leftarrow s'$.

Two learning rates, α_θ and α_w, are predefined for policy and value function parameter updates respectively.

[2]

Table 1: High Level Summary

Value Based	Policy Based
discrete spaces	continuous spaces
guaranteed convergence in restricted cases, worse in practice	better learning properties, but local solutions
bias variance tradeoff, sample efficient	high variance, sample inefficient

Actor Critic attempts to take the best of both worlds

[2] in reality, model-free RL in general has high variance, low sample efficiency, poor convergence properties, especially in high-dimensional stochastic environments with sparse reward signals

4 Exploration Exploitation

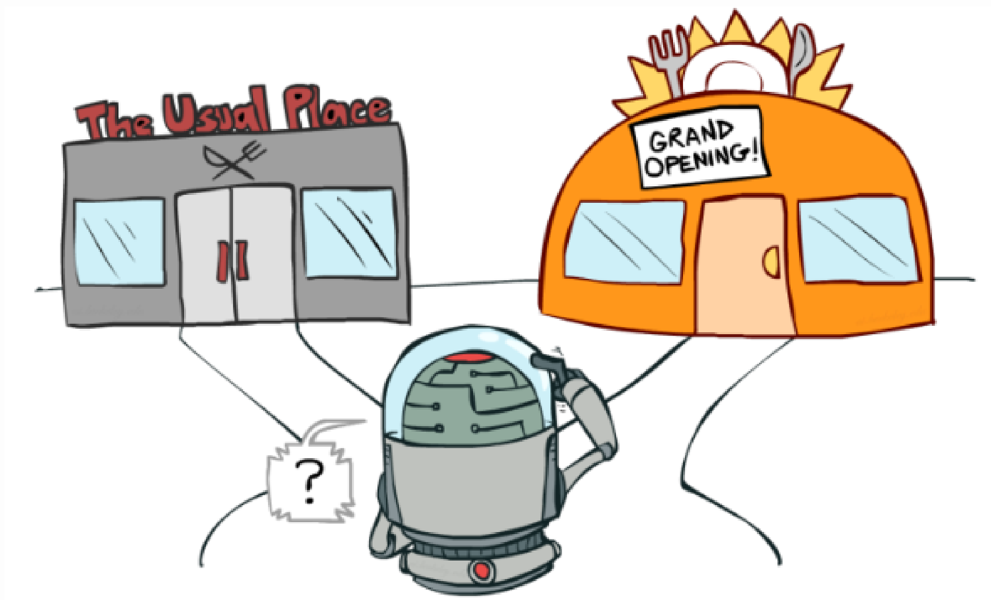


Fig. 1. A real-life example of the exploration vs exploitation dilemma: where to eat? (Image source: UC Berkeley AI course slide, lecture 11.)

- in the presence of *incomplete information* and *stochastic environments*, balance between choosing locally sub-optimal decisions in the interest of gathering valuable information, and exploiting known information

4.1 Bernoulli Multi-Armed Bandits

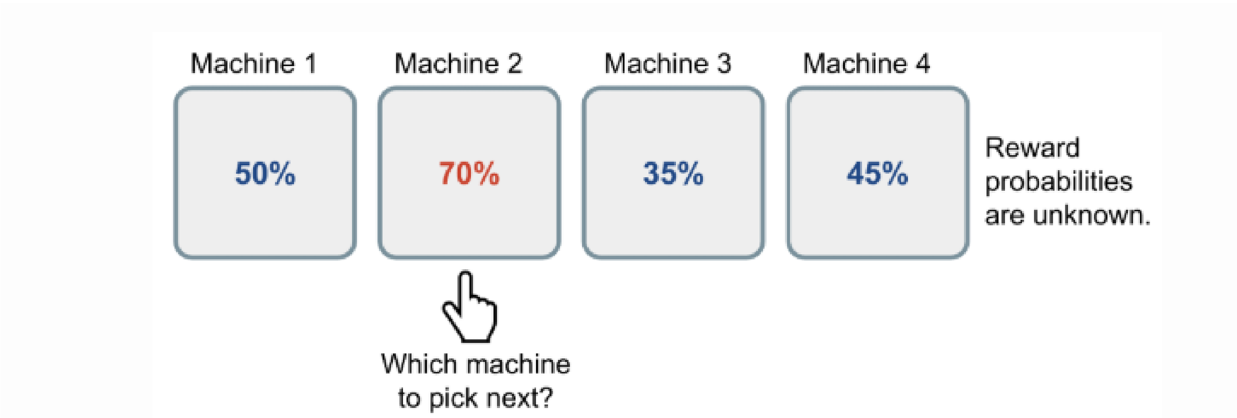
A tuple $\langle \mathcal{A}, \mathcal{R}, \rangle$ with:

- K machines with unkown reward probabilities: $\{\theta_1, ..., \theta_K\}$
- $\mathcal{A}$ is set of actions: one for each machine
- $\mathcal{R}$ is reward probability

At each time step, we choose one of the machine $a \in \mathcal{A}$, and we observe reward:

$$r_t = \mathcal{R}(a_t) = \begin{cases} 1 & \textit{with probability } \theta_a \\ 0 & \textit{otherwise} \end{cases} \tag{38}$$

Goal: maximize cumulative reward over some length of time



4.2 ε-greedy

$$a = \begin{cases} argmax_{a \in \mathcal{A}} Q(a) & \textit{with probability } 1-\varepsilon \\ \sim \textit{Unif}(\mathcal{A}) & \textit{with probability } \varepsilon \end{cases} \tag{39}$$

- take the best known action most of the time, and occasionally do random exploration
- could end up exploring bad action many times
- linear regret, but can be made sublinear using decaying schedules

4.3 Upper Confidence Bounds

Idea: optimism in the face of uncertainty (quantify uncertainty and explore actions with strong potential to have an optimal value)

- Let $\mathcal{U}(a)$ be an upper bound of the true reward value, which is a function of the number of trials.
- Select greediest actions to maximize the upper bound:

$$a_{UCB} = argmax_{a\ in \mathcal{A}}(Q(a) + \mathcal{U}(a)) \tag{40}$$

- estimate the upper confidence bound using *Hoeffding Inequality*:

4.4 Thompson Sampling

- assume functional form and prior on reward distribution, then do bayesian inference to compute posterior over probability that an action is optimal

$$\begin{aligned} \pi(a|h_t) &= \mathbb{P}[Q(a) > Q(a'), \forall a' \neq a | h_t] \quad (41) \\ &= \mathbb{E}_{\mathcal{R}|h_t}[\mathbb{1}[a = argmax_{a \in \mathcal{A}} Q(a)]] \quad (42) \end{aligned}$$

where h_t is the history/trajectory at time t

- at each timestep, sample expected reward from prior for every action
- greedily select best action from the samples
- compute posterior given prior and likelihood and repeat

Problem?

- In practice, posterior inference is intractable, and we result to approximation of the posterior

5 Image Credit

- `https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html`
- `https://lilianweng.github.io/lil-log/2018/01/23/the-multi-armed-bandit-problem-and-its-solutions.html`