

Course 1 - Fundamentals of Reinforcement Learning

Module 2

New terms:

short/long-term reward

policies

planning methods

dynamic programming

reward

time steps

Video: Sequential Decision Making with Evaluative Feedback

Action-Value function

- Giá trị của hành động (q_*) là giá trị kỳ vọng của tất cả các giá trị khả thi khi thực hiện hành động a

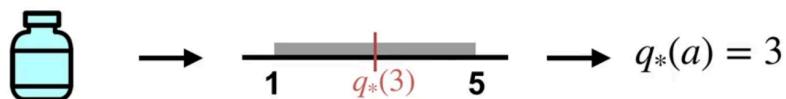
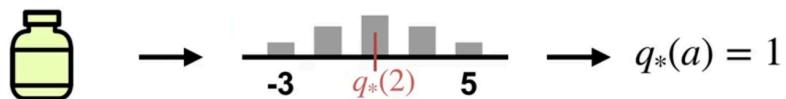
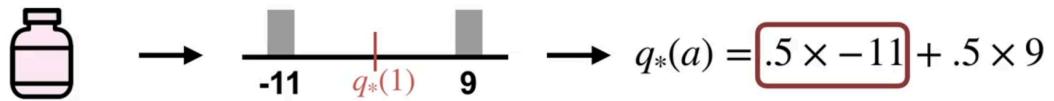
$$\begin{aligned} q_*(a) &\doteq \mathbb{E}[R_t \mid A_t = a] \quad \forall a \in \{1, \dots, k\} \\ &= \sum_r p(r \mid a) r \end{aligned}$$

Giá trị của hành động q là số chưa biết -> cần được ước tính! q_* : giá trị kỳ vọng thực sự q : giá trị kỳ vọng ước tính

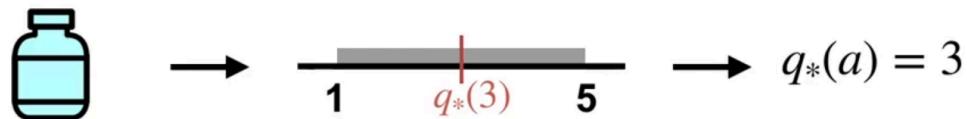
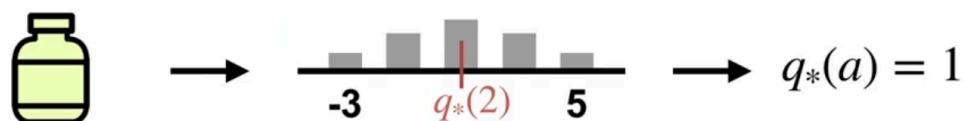
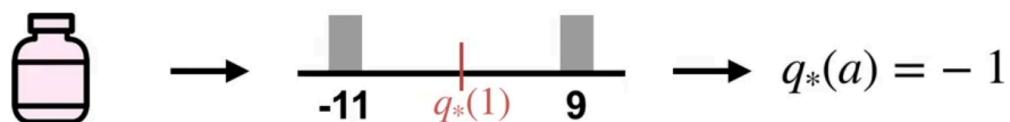
- Mục tiêu là chọn hành động a để tối đa hóa phần thưởng/giá trị kỳ vọng của hành động

$$\arg \max_a q_*(a)$$

Calculating $q_*(a)$



Calculating $q_*(a)$



How is the bandit problem similar to or different from the supervised learning problem?

Vietnamese

Giống: cả 2 đều có mục tiêu đạt được kết quả tối ưu được đo lường bởi 1 hàm số (supervised: loss function, bandit problem: q^* /reward function), supervised được train trên 1 tập data hữu hạn và cố định, bandit problem thì có số lượng action là 1 tập hữu hạn các action (K).

Khác: supervised learning tối đa hóa hàm mất mát trên 1 tập data cố định, ko thay đổi, label là cố định; bandit problem thì có label là giá trị kỳ vọng trên 1 phân phối xác suất.

English

Similarities

1. Optimization Objective

- Both aim to optimize a measurable function:
 - *Supervised Learning*: Minimizes a **loss function** (e.g., cross-entropy, MSE).
 - *Bandit Problems*: Maximizes a **reward function** (e.g., Q*-value, expected reward).

2. Finite Action Space

- Supervised learning uses a fixed, finite dataset.
- Bandit problems assume a finite set of **K actions** (e.g., choosing between K ad variants).

Key Differences

Aspect	Supervised Learning	Bandit Problems
Data Dynamics	Static dataset with fixed labels	Dynamic, stochastic rewards from a distribution
Feedback Type	Full feedback (labels for all inputs)	Partial feedback (reward only for chosen action)
Exploration Strategy	No exploration needed (deterministic training)	Requires exploration-exploitation trade-off (e.g., ϵ -greedy, UCB)
Objective	Generalize to unseen data	Maximize cumulative reward over interactions

Video: Learning Action Values

Sample-Average Method

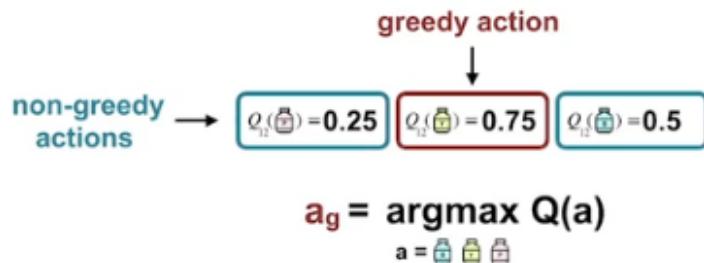
$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t}$$

$$= \frac{\sum_{i=1}^{t-1} R_i}{t-1}$$

Greedy action

Method of choosing action: choosing the **greedy action** a.k.a the action currently has the largest estimated value

Action Selection



Video: Estimating Action Values Incrementally

Incremental update rule

Incremental update rule

Recall

$$Q_n = \frac{1}{n-1} \sum_{i=1}^{n-1} R_i$$

$$\begin{aligned} Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n R_i \\ &= \frac{1}{n} (R_n + \sum_{i=1}^{n-1} R_i) && \xrightarrow{\quad} \quad = \frac{1}{n} (R_n + (n-1)Q_n) \\ &= \frac{1}{n} (R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i) && = \frac{1}{n} (R_n + nQ_n - Q_n) \\ & && = Q_n + \frac{1}{n} (R_n - Q_n) \end{aligned}$$

$$NewEstimate \leftarrow OldEstimate + StepSize [Target - OldEstimate]$$

$$Q_{n+1} = Q_n + \alpha_n (R_n - Q_n)$$

$$\alpha_n \rightarrow [0, 1]$$

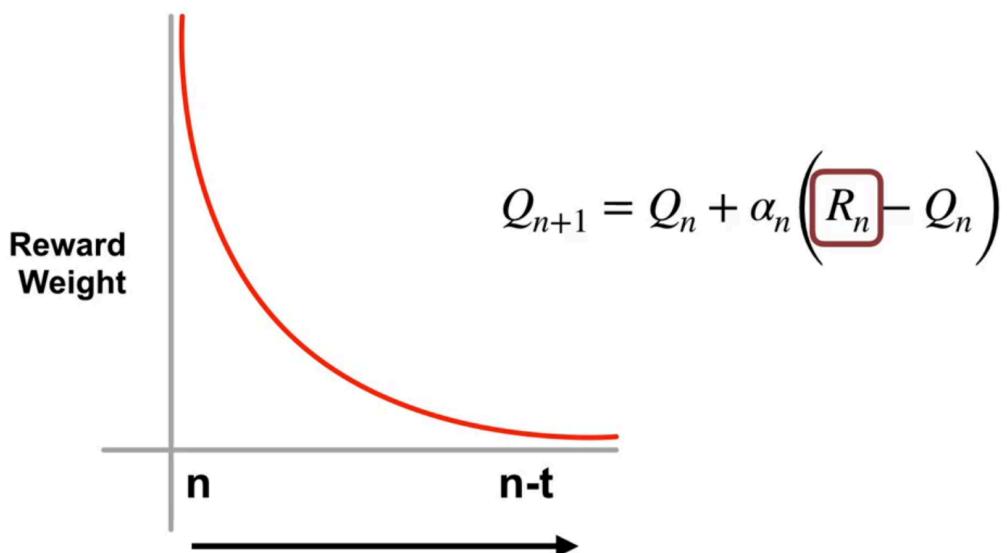
Sample average method

$$\alpha_n = \frac{1}{n}$$

α (step size), là 1 hằng số

Non-stationary bandit problem (rewad ko cố định và có thể thay đổi theo thời gian)

Trong các bài toán non-stationary (có tính thay đổi theo thời gian, học trực tuyến a.k.a online learning), stepsize là 1 hằng số cố định sẽ giúp estimate và điều chỉnh q tốt và nhanh hơn là 1 step size giảm dần theo thời gian (decaying: $\frac{1}{n}$)



the most recent rewards affect the estimate more than older rewards (các reward mới nhất ảnh hưởng đến giá trị ước lượng hơn các reward ở các bước xa hơn)

Decaying past rewards

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha_n (R_n - Q_n) \\
&= \alpha_n R_n + Q_n - \alpha_n Q_n \\
&= \alpha_n R_n + (1 - \alpha_n) Q_n \\
&= \alpha_n R_n + (1 - \alpha_n) [\alpha_n R_{n-1} + (1 - \alpha_n) Q_{n-1}] \\
&= \alpha_n R_n + (1 - \alpha_n) \alpha_n R_{n-1} + (1 - \alpha_n)^2 Q_{n-1} \\
&= \alpha_n R_n + (1 - \alpha_n) \alpha_n R_{n-1} + (1 - \alpha_n)^2 \alpha_n R_{n-2} + \dots \\
&\quad + (1 - \alpha_n)^{n-1} \alpha_n R_1 + (1 - \alpha_n)^n Q_1 \\
&= (1 - \alpha)^n Q_1 + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} R_i
\end{aligned}$$

$Q_1 \rightarrow \text{initial action-value}$

Đóng góp của Q_1 với giá trị ước tính tại bước $n + 1$ giảm dần theo cấp lũy thừa theo thời gian, các giá trị reward ở các bước cũ đóng góp theo cấp lũy thừa ít hơn. Sự ảnh hưởng của giá trị khởi tạo (Q_1) tiến gần về 0 khi càng có thêm nhiều data, các giá trị mới nhất quyết định giá trị ước tính hiện tại (Q_{n+1})

Exploration vs. Exploitation Tradeoff

Video: What is the trade-off?

Exploration and Exploitation

- Exploration - improve knowledge for long-term benefit
- Exploitation - exploit knowledge for short-term benefit (being greedy w.r.t estimated values, may not actually get the most reward)
- Round Robin fashion: tuần tự theo chu kỳ

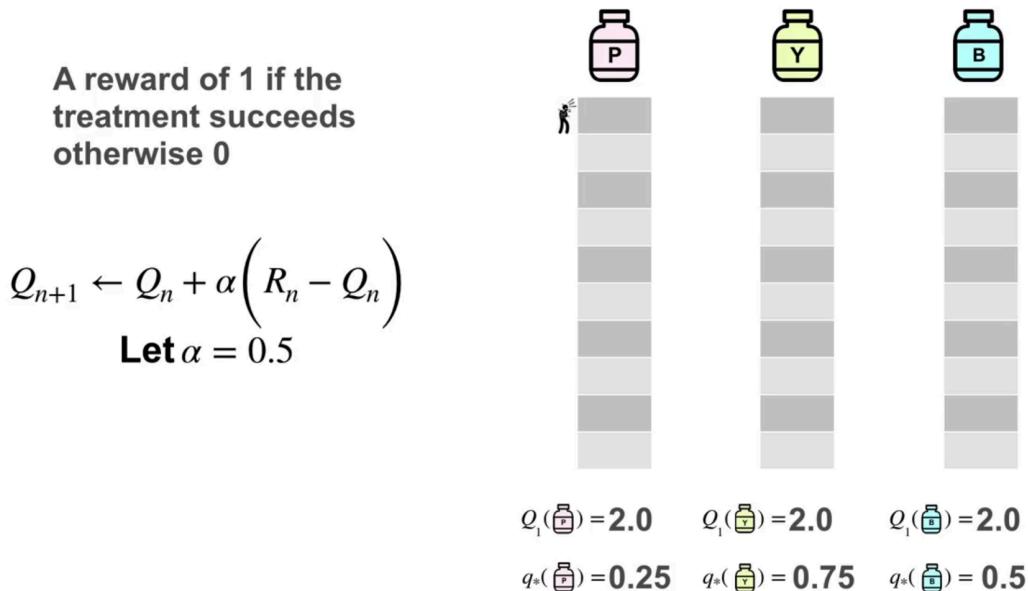
Các phương pháp để cân bằng giữa Exploration và Exploitation

Epsilon-Greedy Action Selection

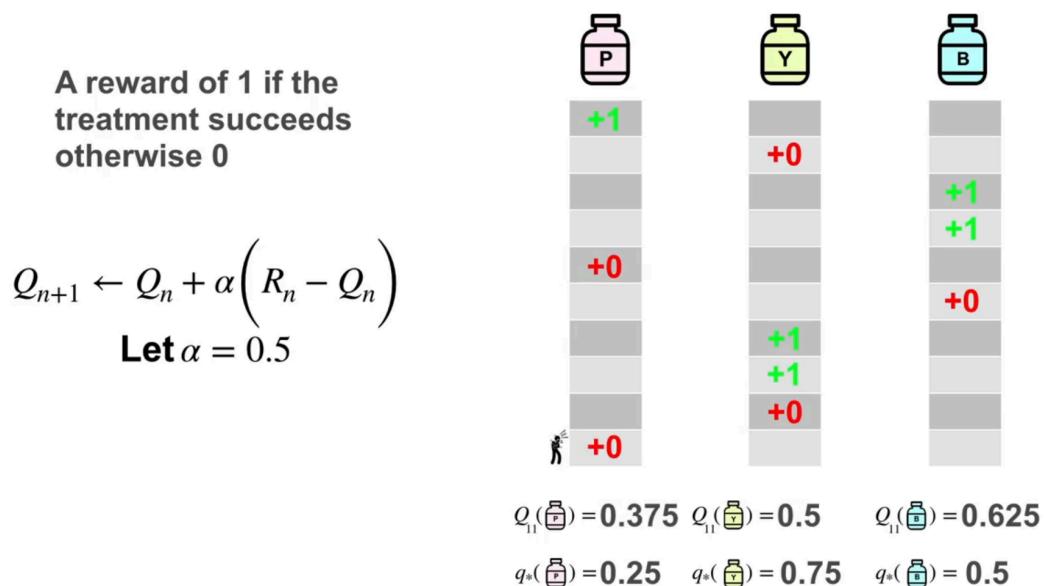
$$A_t \leftarrow \begin{cases} \arg \max_a Q_t(a) & \text{with probability } 1 - \epsilon \\ a \sim \text{Uniform}(\{a_1, \dots, a_k\}) & \text{with probability } \epsilon \end{cases}$$

Lựa chọn khám phá (explore) ngẫu nhiên 1 hành động với xác suất ϵ từ xác suất phân phối đều, và lựa chọn greedy hành động có ước tính phần thưởng Q_t lớn nhất trong số các hành động (exploit)

Video: Optimistic Initial Values



Khởi tạo giá trị kỳ vọng ước lượng khởi đầu cao và thực hiện chiến thuật lựa chọn tham lam (greedy selection) giúp agent có thể explore các lựa chọn khác nhau ở các timestep đầu tiên và update dần về giá trị hành động thực tế

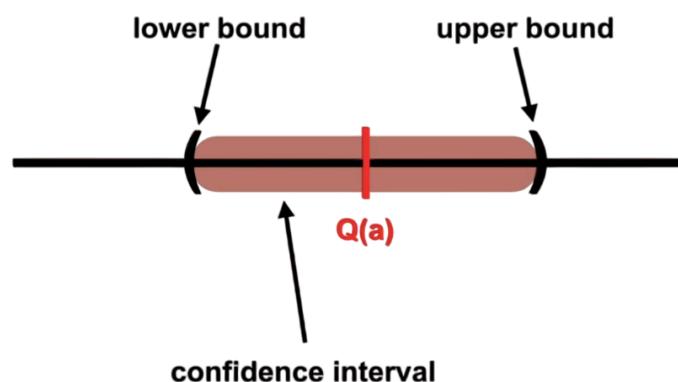


Giới hạn:

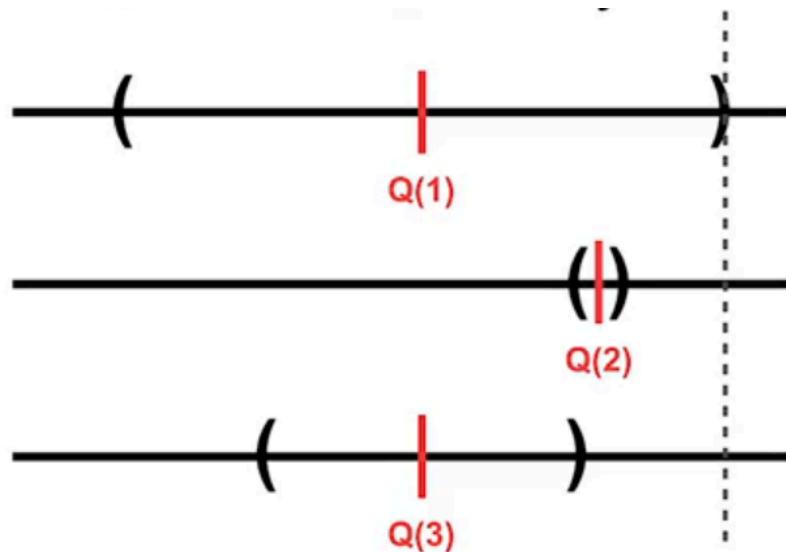
- Chỉ explore ở các bước đầu tiên, sau khi đến bước nào đó sẽ dừng explore
- Không phù hợp với các bài toán có reward thay đổi theo thời gian (non-stationary problems)
- Không thể biết được giá trị khởi đầu lắc quan sát để là bao nhiêu (vì không biết giá trị tối đa của reward)

Video: Upper-Confidence Bound (UCB) Action Selection

Uncertainty in Estimates



Optimism in the Face of Uncertainty



Upper-Confidence Bound (UCB) Action Selection

chọn action có cận trên của giá trị hành động là cao nhất

$$A_t \doteq \arg \max_a [Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}}]$$

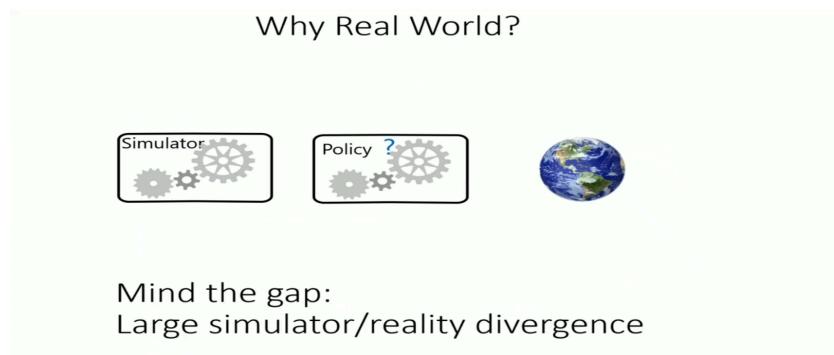
$c \sqrt{\frac{\ln t}{N_t(a)}}$ is upper-confidence bound (UCB) exploration term

c is user-specified parameter that controls the amount of exploration

$$A_t \doteq \operatorname{argmax} \left[Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right]$$

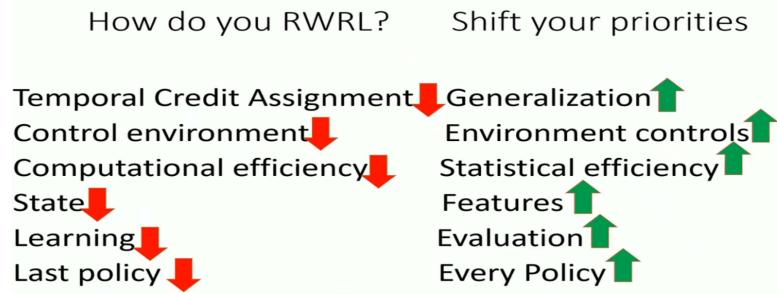
Exploit Explore

Video: Jonathan Langford: Contextual Bandits for Real World Reinforcement Learning

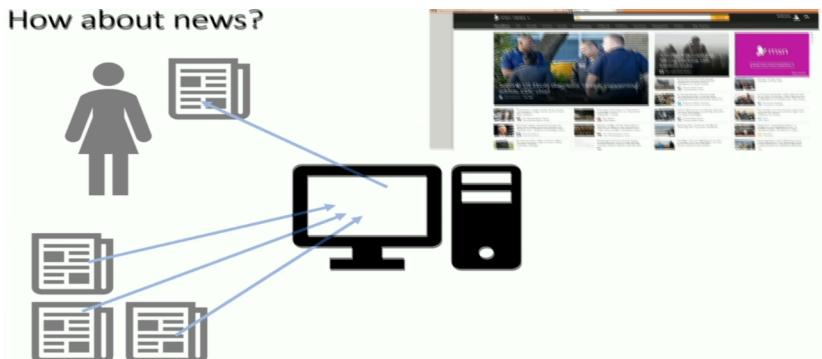


There's a gap between the simulator and the reality.

Real World Reinforcement Learning



An example: Contextual Bandits



Having a set of possible news articles in interest today, suggest news article for new user come to website that they maybe have interest in, get feedback about whether or not they actually read the article.

Repeatedly:

1. Observe features x (user geolocation, past profile behaviour, features of available actions,...)
2. Choose action $a \in A$
3. Observe reward r

Goal: Maximize reward

Major caveat: No credit assignment (for selected action, what's the reward if we had chosen a different action?)

Video: Week 1 Summary

Exploration vs Exploitation strategies

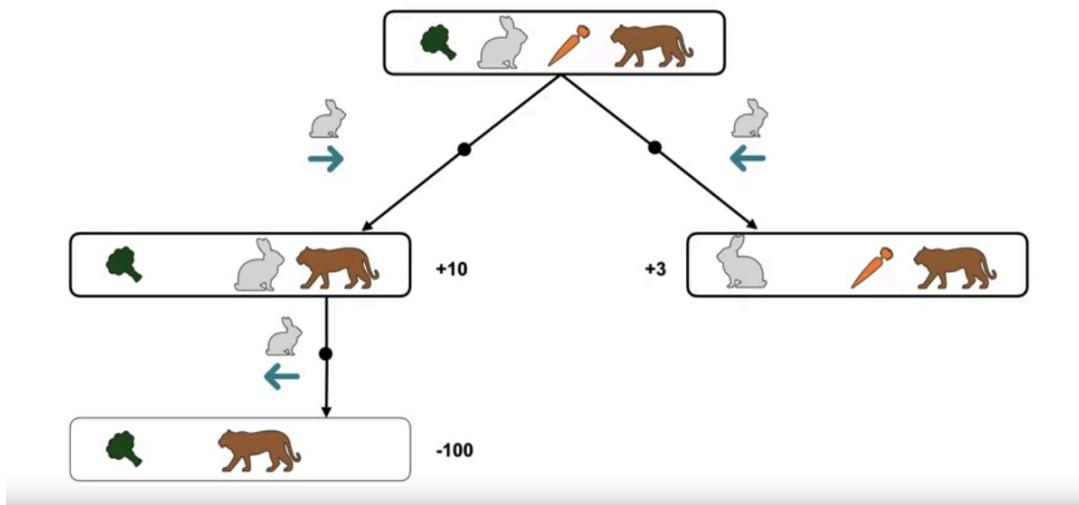
- ϵ -Greedy Action Selection
- Optimistic Initial Values
- Upper-Confidence Bound (UCB) Action Selection

Module 3

Introduction to Markov Decision Processes

Video: Markov Decision Processes

- Các tình huống khác nhau sẽ xuất hiện các lựa chọn hành động khác nhau
- Hành động dẫn đến trạng thái của thế giới thay đổi action -> state



Markov Decision Processes (MDPs) là các khung toán học được sử dụng để mô hình hóa các vấn đề ra quyết định, trong đó một tác nhân tương tác với một môi trường trong một chuỗi các bước thời gian riêng biệt (discrete time steps).

Markov Decision Processes terms:

- **Agent:** là thực thể mà chúng ta đang huấn luyện để đưa ra các quyết định chính xác.
- **Environment:** môi trường xung quanh mà agent tương tác.
- **State:** (thông tin) trạng thái hiện tại của agent trong môi trường để đưa ra quyết định.
- **Action:** lựa chọn mà agent thực hiện tại mỗi bước thời gian hiện tại.
- **Policy:** Policy là quá trình suy nghĩ hoặc chiến lược đứng sau việc lựa chọn một action.
- **P (Transition Probability):** $P(s' | s, a)$ cho biết xác suất chuyển sang trạng thái s' khi thực hiện hành động a tại trạng thái s . Điều này thể hiện tính ngẫu nhiên của môi trường.
- **R (Reward Function):** $R(s, a, s')$ xác định phần thưởng nhận được ngay lập tức sau khi chuyển từ trạng thái s sang trạng thái s' do thực

hiện hành động a .

Sometimes:

- γ (Discount Factor): Một giá trị trong khoảng từ 0 đến 1, quyết định mức độ quan trọng của các phần thưởng trong tương lai.

How MDPs Work

Ở mỗi bước thời gian:

- Agent quan sát trạng thái hiện tại.
- Chọn một action dựa trên policy (một ánh xạ từ state đến action).
- Environment chuyển sang một state mới dựa trên xác suất chuyển tiếp (transition probabilities).
- Agent nhận được reward tương ứng với action đã thực hiện.

Mục tiêu là tìm ra một **chính sách tối ưu** để tối đa hóa tổng phần thưởng kỳ vọng (expected cumulative reward) theo thời gian.

The dynamics of an MDP (Markov property)

Khi agent thực hiện hành động a trong trạng thái s , sẽ có nhiều trạng thái kế tiếp s' và phần thưởng r có thể xảy ra. Hàm động lực chuyển tiếp $P(s', r|s, a)$ mô tả xác suất của từng kết quả (s', r) .

Công thức:

$$P(s', r|s, a) = P(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$$

$p(s', r|s, a)$ là xác suất xảy ra trạng thái s' và nhận được phần thưởng r với hành động a từ trạng thái s

Giải thích:

- Hàm này định nghĩa "luật chơi" của môi trường:
 - **Đầu vào:** Trạng thái hiện tại s + hành động a .
 - **Đầu ra:** Phân phối xác suất cho tất cả cặp (s', r) có thể.
- Ví dụ: Robot di chuyển nhưng có 10% khả năng trượt (sang trạng thái khác ngoài dự định).

Tính chất:

- $\sum_{s'} P(s', r | s, a) = 1$ (tổng xác suất bằng 1)

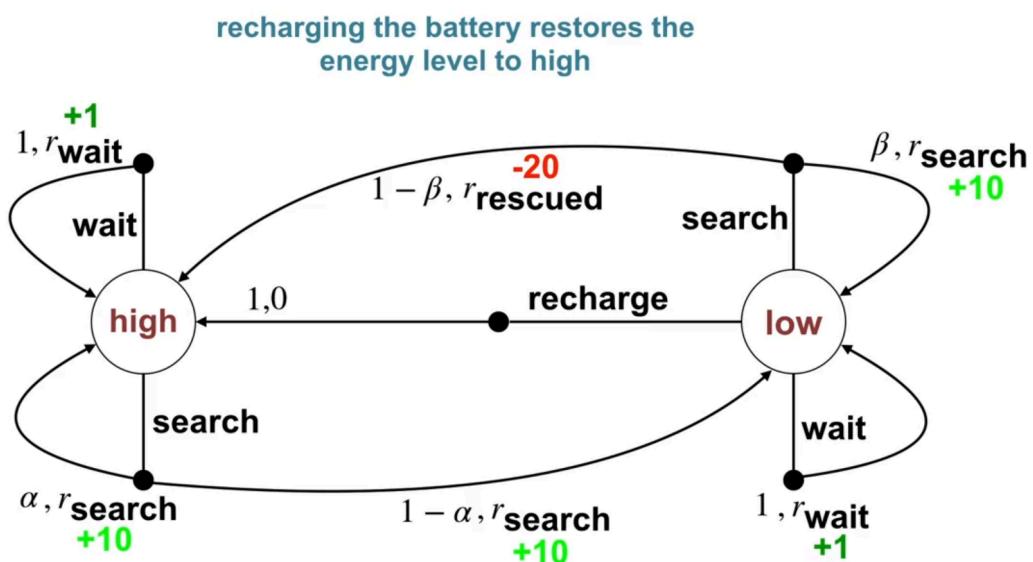
$$p : S \times R \times S \times A \rightarrow [0, 1]$$

$$\sum_{s' \in S} \sum_{r \in R} p(s', r | s, a) = 1, \forall s \in S, a \in A(s)$$

- Môi trường deterministic là trường hợp đặc biệt: $P(s', r | s, a) = 1$ cho một cặp (s', r) duy nhất.
- Tính chất Markov: trạng thái tương lai chỉ phụ thuộc vào trạng thái và hành động hiện tại, không phải vào chuỗi các sự kiện trước nó (ghi nhớ về các trạng thái trước đó không giúp cải thiện cho dự đoán về tương lai)

Video: Examples of MDPs

Dynamics of the Recycling Robot



state, action có thể đơn giản, chi tiết (low-level) hoặc abstract (high-level)



Task: The goal of the robot is to pick-and-place objects

State: latest readings of joint angles and velocities

Action: the amount of voltage applied to each motor

Reward: +100 when an object is successfully placed
-1 for each unit of energy consumed

Goal of Reinforcement Learning

Video: The Goal of Reinforcement Learning

Goal of an Agent: Formal definition

Mục tiêu của agent là tối đa hóa tổng phần thưởng nhận về (return a.k.a giá trị tích lũy) trong tương lai

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots$$

G_t là 1 biến ngẫu nhiên

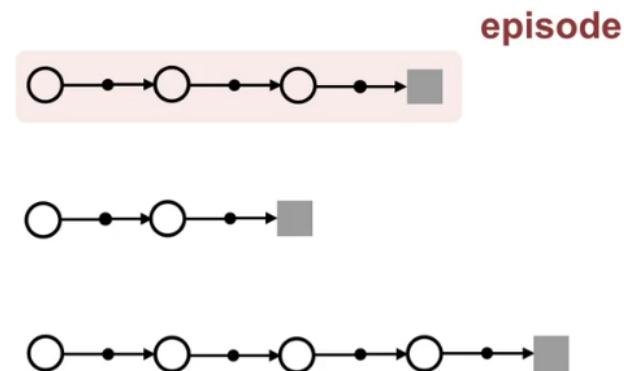
$$\text{E}[G_t] = \text{E}[R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T]$$

maximize the expected return

Tối đa hóa giá trị mong đợi cho phần thưởng nhận được trong tương lai (tổng các phần thưởng phải là 1 số hữu hạn)

T là thời điểm kết thúc (final timestep)

Epsodic Tasks (tập hành động)



Các tương tác được chia thành các đoạn nhỏ gọi là các episode, mỗi episode bắt đầu độc lập với kết thúc của episode trước đó, khi 1 episode kết thúc, agent được đặt lại về trạng thái bắt đầu, mỗi episode sẽ có 1 trạng thái kết thúc.

Ví dụ:

- Chơi một ván cờ (mỗi ván là một episode) -> trạng thái kết thúc: thắng, hòa, đầu hàng
- Một lần robot di chuyển từ vị trí xuất phát đến đích

Discussion Prompt: Is the reward hypothesis sufficient?

- Multi-objective tasks: Can't balance goals like speed and safety in self-driving cars.
- Risk-sensitive tasks: Ignores risk, e.g., investors want to avoid rare catastrophic losses, not just maximize average return.
- Modal/counterfactual tasks: Can't reason about what could have happened, e.g., a robot must maintain the ability to avoid collisions.
- Dynamic/subjective preferences: Can't adapt to changing or conflicting values, e.g., recommenders may need to shift from engagement to well-being.
- Non-Markovian dependencies: Can't encode history-based requirements, e.g., loan systems must ensure fairness over time.

Video: Michael Littman: The Reward Hypothesis

Whence Behavior?

- give a man a fish and he'll eat for a day

- Programming (GOFAI - Good-old fashioned AI)
- teach a man to fish and he'll eat for a lifetime
 - Examples (LfD)
- give a man a taste for fish and he'll figure out how to get fish, even if the details change!
 - Optimization (RL)

Goals as Rewards

- 1 for goal, 0 otherwise
 - goal-reward representation
- -1 for not goal, 0 once goal reached
 - action-penalty representation

Whence Rewards?

- Programming
 - Coding
 - Human-in-the-loop (con người hay thay đổi câu trả lời của mình tùy theo cách mà agent đang học - non-stationary reward)
- Examples
 - Mimic reward (đưa ra ví dụ về phần thưởng bởi ví dụ, agent sẽ học cách "sao chép" phần thưởng được cung cấp, reward -> behavior)
 - Inverse reinforcement learning (trainer cho agent học bằng việc cung cấp ví dụ về "hành vi" mong muốn để agent tự suy luận ra reward mục tiêu cần đạt được tối ưu tương đương cho "hành vi" được cung cấp, behavior -> reward)
- Optimization
 - Evolutionary optimization: cho nhiều model cùng học và model nào "sống sót" (có kết quả và thuật toán tốt hơn 1 người) thì sẽ được giữ lại cho các bước học tiếp theo

- o Meta RL

Challenges to the Hypothesis (Những thách thức của giả thuyết phần thưởng)

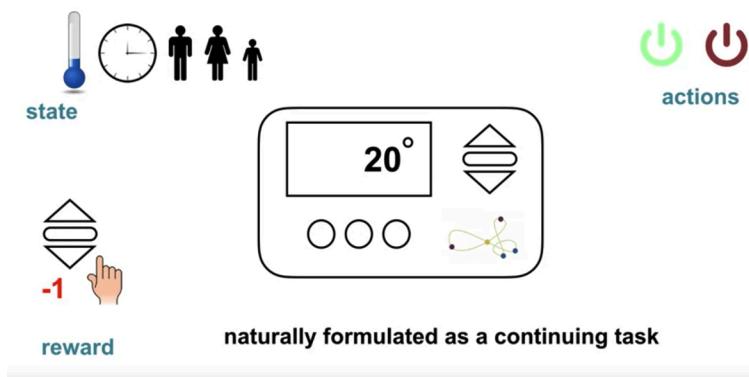
- Mục tiêu không phải là giá trị kỳ vọng của phần thưởng tích lũy
 - o Risk-averse problem (giảm thiểu rủi ro): chọn kết quả tốt nhất mà ít khả năng có biến cố xảy ra nhất, có thể mô hình hóa bằng việc khuếch đại các kết quả tiêu cực trong phần thưởng
 - o Cân bằng đa dạng hóa
- Có thể tương thích với cách học như của con người không?
 - o Theo đuổi 1 nhiệm vụ mù quáng không phải luôn luôn tốt
 - o Mục tiêu có thể thay đổi, phát triển theo thời gian

Continuing Tasks

Video: Continuing Tasks

Episodic Tasks	Continuing Tasks
Interaction breaks naturally into episodes	Interaction goes on continually
Each episode ends in a terminal state	No terminal state
Episodes are independent	Results are dependent
$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T$	$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots$

Example: Smart thermostat which regulates the temperature of a building



Discounting

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{k-1} R_{t+k} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned}$$

G_t luôn là hữu hạn khi $0 \leq \gamma < 1$, γ là tỷ số chiết khấu

Chứng minh rằng G_t hữu hạn khi $0 \leq \gamma < 1$

Giả sử R_{\max} là phần thưởng tối đa mà agent có thể nhận được

$$\begin{aligned} G_t &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \sum_{k=0}^{\infty} \gamma^k R_{\max} = R_{\max} \sum_{k=0}^{\infty} \gamma^k \\ &= R_{\max} \times \frac{1}{1-\gamma} \end{aligned}$$

Mà $\sum_{k=0}^{\infty} \gamma^k$ là tổng của 1 chuỗi hình học hội tụ (chuỗi có tổng tiến tới một giá trị hữu hạn) khi $\gamma < 1$ có tổng hội tụ về $\frac{1}{1-\gamma}$.

$\Rightarrow R_{\max} \times \frac{1}{1-\gamma}$ là hữu hạn và là cận trên của G_t

$\Rightarrow G_t$ là hữu hạn

Effect of γ on agent behavior

- $\gamma = 0$

$$\begin{aligned} G_t &\doteq R_{t+1} + \cancel{\gamma} R_{t+2} + \cancel{\gamma^2} R_{t+3} + \dots + \cancel{\gamma^{k-1}} R_{t+k} + \dots \\ &= R_{t+1} + \cancel{0} R_{t+2} + \cancel{0^2} R_{t+3} + \dots + \cancel{0^{k-1}} R_{t+k} + \dots \\ &= R_{t+1} \end{aligned}$$

Khi $\gamma = 0$, giá trị tích lũy G_t sẽ chỉ đơn giản là reward ở bước tiếp theo, agent sẽ có tầm nhìn ngắn hạn và chỉ quan tâm đến phần thưởng mong đợi tức thời

- $\gamma \rightarrow 1$

$$\begin{aligned} G_t &\doteq R_{t+1} + \cancel{\gamma} R_{t+2} + \cancel{\gamma^2} R_{t+3} + \dots + \cancel{\gamma^{k-1}} R_{t+k} + \dots \\ &\approx R_{t+1} + \cancel{1} R_{t+2} + \cancel{1^2} R_{t+3} + \dots + \cancel{1^{k-1}} R_{t+k} + \dots \\ &= R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_{t+k} + \dots \end{aligned}$$

Phần thưởng tức thời và trong tương lai có độ quan trọng gần bằng nhau
nên agent sẽ có tầm nhìn xa hơn

Recursive nature of returns (tính đệ quy của giá trị tích lũy)

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ G_t &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

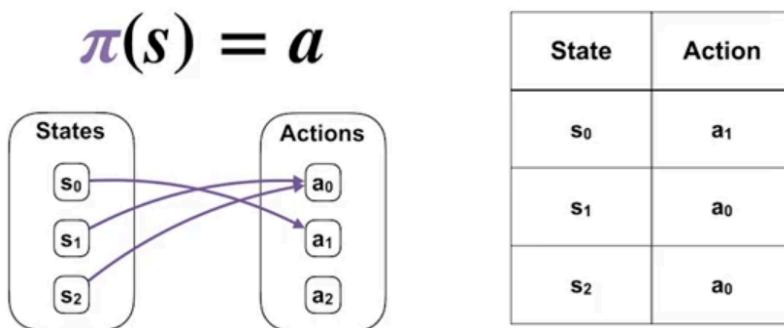
Module 4

Policies and Value Functions

Video: Specifying Policies

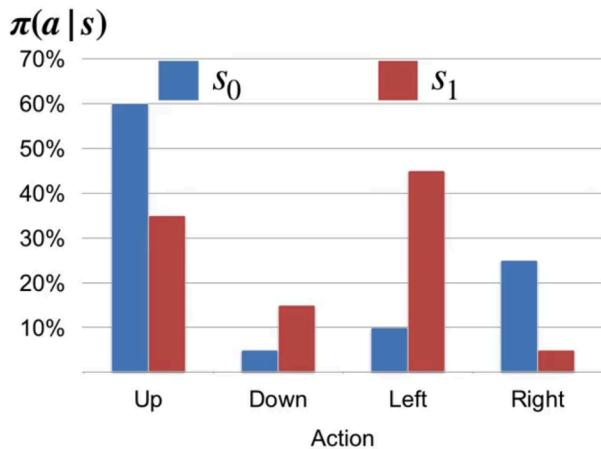
Deterministic Policy (chính sách xác định)

π (policy: chính sách) là phương pháp lựa chọn (ánh xạ) hành động a từ trạng thái s .



Stochastic Policy (chính sách ngẫu nhiên)

$\pi(a|s)$ là xác suất lựa chọn hành động a khi đang ở trạng thái s , tổng xác suất của tất cả hành động a trong trạng thái s phải bằng 1 và không có xác suất hành động nào được là số âm.

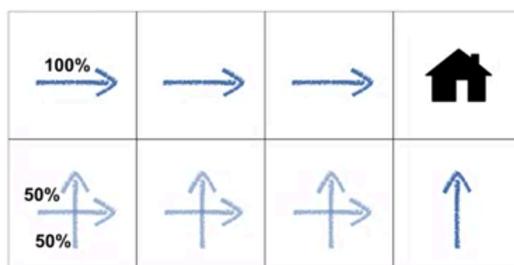


$$\pi(a | s)$$

$$\sum_{a \in \mathcal{A}(s)} \pi(a | s) = 1$$

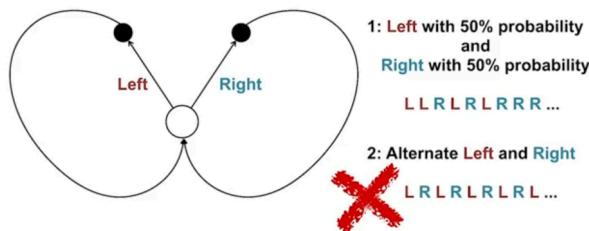
$$\pi(a | s) \geq 0$$

Stochastic Policy Example



Valid and invalid policies

Policy trong MDPs chỉ quyết định hành động tiếp theo dựa trên trạng thái s hiện tại, không phụ thuộc vào các hành động hay trạng thái trước đó. Trong MDPs, trạng thái hiện tại luôn chứa đầy đủ thông tin cho việc ra quyết định cho hành động tiếp theo.



Video: Value Functions

State-value functions

Giá trị tích lũy kỳ vọng khi bắt đầu ở trạng thái s với chính sách π

Recall that

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi} [G_t | S_t = s]$$

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

Action-value functions

Giá trị tích lũy kỳ vọng khi bắt đầu ở trạng thái s , thực hiện hành động a với chính sách π

$$q_{\pi}(s, a) \doteq \mathbb{E}_{\pi} [G_t | S_t = s, A_t = a]$$

Value functions represent how good it is for the agent to be in a given state or take a particular action in a given state under a particular policy.

Value functions are crucial for reinforced learning because they allow the agent to query the quality of the current situations instead of waiting to observe the long-term outcome:

- The return is not immediately available
- The return can be random due to stochasticity in both the policy and environment dynamics

Video: Bellman Equation

State-value Bellman equation

$$\begin{aligned} v_{\pi}(s) &\doteq \mathbb{E}_{\pi}[G_t | S_t = s] \\ &= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s' r} p(s', r | s, a) [r + \gamma \mathbb{E}[G_{t+1} | S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s' r} p(s', r | s, a) [r + \gamma v_{\pi}(s')] \end{aligned}$$

Action-value Bellman equation

$$\begin{aligned}
q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\
&= \sum_{s'} \sum_r p(s', r \mid s, a) [r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']] \\
&= \sum_{s'} \sum_r p(s', r \mid s, a) [r + \gamma \sum_{a'} \pi(a' \mid s') \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s', A_{t+1} = a']] \\
&= \sum_{s'} \sum_r p(s', r \mid s, a) [r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a')]
\end{aligned}$$

Optimality (Optimal Policies & Value Functions)

Video: Optimal Policies

The role of policies in Reinforcement Learning (RL) is pivotal as they dictate the behavior of an agent within its environment.

- Action Selection: Policies determine how an agent selects actions in different states of the environment.
- Learning Objective: Policies define the learning objective for the RL agent.
- Exploration vs. Exploitation: Policies balance exploration and exploitation

Bellman Optimality Equation

An optimal policy π_* is a policy that maximizes the expected cumulative reward over time.

The Bellman optimality equation expresses the optimal value of a state (or state-action pair) in terms of the maximum expected immediate reward plus the discounted value of the successor states.

State-value Bellman optimality function $v_*(s)$

$$v_*(s) = \max_a \sum_{s'} \sum_r p(s', r \mid s, a) [r + \gamma v_*(s')]$$

$v_*(s)$ represents the optimal value of state s under the optimal policy

Action-value Bellman optimality function $q_*(s)$

$$q_*(s, a) = \sum_{s'} \sum_r p(s', r \mid s, a) [r + \gamma \max_{a'} q_*(s', a')]$$

$q_*(s, a)$ represents the optimal value of taking action a in state s under the optimal policy.

Relationship

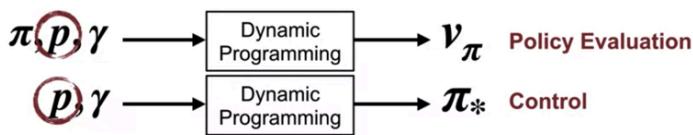
v_* is equal to the maximum of the boxed term over all actions. π_* is the argmax, which simply means the particular action which achieves this maximum.

$$\pi_*(s) = \operatorname{argmax}_a \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma v_*(s')]$$

Module 5

Policy Evaluation (Prediction)

Video: Policy Evaluation vs. Control



- **Policy Evaluation:** Determines the value function of a given policy, assessing how good it is.
- **Policy Control:** Improves the policy to maximize future rewards, finding the optimal strategy.
- **Relative Value:** A policy is considered as good as or better than a policy if the value function under is greater than or equal to the value function under for every state. If there exists at least one state where has a strictly higher value, then is strictly better than .

Video: Iterative Policy Evaluation

```

Input  $\pi$ , the policy to be evaluated
 $V \leftarrow \vec{0}, V' \leftarrow \vec{0}$ 
Loop:
   $\Delta \leftarrow 0$ 
  Loop for each  $s \in \mathcal{S}$  :
     $V'(s) \leftarrow \sum_a \pi(a | s) \sum_{s',r} p(s',r | s,a) [r + \gamma V(s')]$ 
     $\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$ 
   $V \leftarrow V'$ 
until  $\Delta < \theta$  (a small positive number)
Output  $V \approx v_\pi$ 

```

The **dynamics of the environment** in reinforcement learning refer to how the agent interacts with the environment and how the environment responds. This is typically modeled as a **Markov Decision Process (MDP)** with the following components:

1. **States (S)**: The possible situations the agent can be in.
2. **Actions (A)**: The choices available to the agent in each state.
3. **Transition Probabilities ($P(s' | s, a)$)**: The probability of moving to a new state given the current state and action .
4. **Reward Function ($R(s, a, s')$)**: The numerical feedback given to the agent when transitioning between states due to an action.
5. **Policy ($\pi(a | s)$)**: The strategy the agent follows when choosing actions.

Policy Iteration (Control)

We can find the optimal policy by choosing the Greedy action. The Greedy action maximizes the Bellman's optimality equation in each state.

Recall that

Greedy action

$$\pi_*(s) = \operatorname{argmax}_a \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma v_*(s')]$$

?

$$\operatorname{argmax}_a \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma v_\pi(s')]$$

$$\pi(s) = \operatorname{argmax}_a \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma v_\pi(s')] \text{ for all } s \in \mathcal{S}$$

$\rightarrow v_\pi$ obeys the Bellman optimality equation $\rightarrow \pi$ is optimal

$$\begin{aligned}
 q_{\pi'}(s, \pi'(s)) &\geq q_\pi(s, \pi(s)) \quad \text{for all } s \in S & \rightarrow \pi' \geq \pi \\
 q_{\pi'}(s, \pi'(s)) &> q_\pi(s, \pi(s)) \quad \text{for at least one } s \in S & \rightarrow \pi' > \pi
 \end{aligned}$$

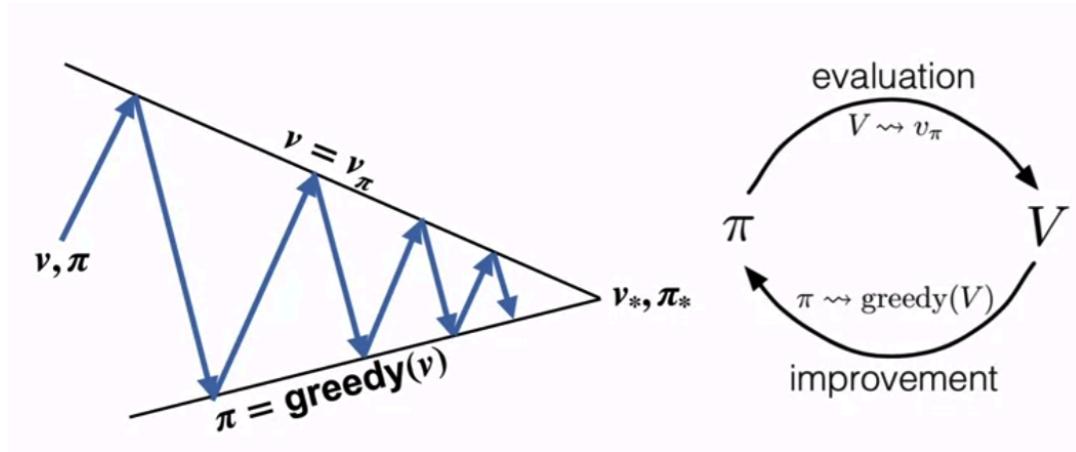
Suppose you have two policies and their value functions for all states:

- If $v_{\pi_2}(s) \geq v_{\pi_1}(s)$ for every s , π_2 is at least as good as π_1 .
- If for some state s^* , $v_{\pi_2}(s^*) > v_{\pi_1}(s^*)$, then π_2 is **strictly** better.

Policy Iteration

Policy iteration consists of two distinct steps: evaluation and improvement.

1. **Evaluation Step:** We first evaluate our current policy π_1 , which gives us a new value function v_{π_1} that accurately reflects the value of π_1 .
2. **Improvement Step:** The improvement step then uses v_{π_1} to produce a greedy policy π_2 . At this point, π_2 is greedy with respect to the value function of π_1 , but v_{π_1} no longer accurately reflects the value of π_2 , so we need to re-calculate again.



Policy Iteration pseudo code using iterative policy evaluation for estimate $\pi \sim \pi^*$

```

1. Initialization
 $V(s) \in \mathbb{R}$  and  $\pi(s) \in \mathcal{A}(s)$  arbitrarily for all  $s \in \mathcal{S}$ 

2. Policy Evaluation
Loop:
 $\Delta \leftarrow 0$ 
Loop for each  $s \in \mathcal{S}$ :
 $v \leftarrow V(s)$ 
 $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$ 
 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number determining the accuracy of estimation)

3. Policy Improvement
 $policy-stable \leftarrow true$ 
For each  $s \in \mathcal{S}$ :
 $old-action \leftarrow \pi(s)$ 
 $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ 
If  $old-action \neq \pi(s)$ , then  $policy-stable \leftarrow false$ 
If  $policy-stable$ , then stop and return  $V \approx v_*$  and  $\pi \approx \pi_*$ ; else go to 2

```

Generalized Policy Iteration (GPI)

Value Iteration

Value Iteration for estimating $p \sim p^*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

- | $\Delta \leftarrow 0$
- | Loop for each $s \in \mathcal{S}$:
- | $v \leftarrow V(s)$
- | $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$
- | $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
 $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

Alternatives

- **Monte Carlo method:** Estimates state values by averaging multiple returns, but requires many samples due to randomness.
- **Brute-force search:** Tries every possible deterministic policy to find the best one, but is extremely slow due to exponential complexity.

Flexibility of the Policy Iteration Framework

Asynchronous dynamic programming approaches must ensure all states get updated at some point to maintain accuracy.