# Fundamentals

**K-armed Bandit Problem**

- Action-Values
  - Sample-average method
  - Greedy action
- Non-stationary Bandit Problems
  - Incremental update rule
- Exploration-Exploitation Dilemma
  - Epsilon-Greedy action selection
  - Optimistic initial values
  - UCB action selection
- Contextual Bandits

**Markov Decision Processes**

**Goal of RL**

- Reward hypothesis
- Episodic tasks
- Continuing tasks

**Policies & Optimality**

- Policy & Value functions
- Bellman Equation
- Optimality
  - Brute-Force Search
  - Bellman Optimality Equation

**Dynamic Programming**

- Policy Evaluation (Prediction)
- Policy Iteration (Control)
- Generalized Policy Iteration
  - Value Iteration
  - Monte-Carlo Method vs Bootstrapping
  - Brute-Force Search vs Policy Improvement
  - Curse of Dimensionality

# K-armed Bandit Problem

- <u>Definition</u>: An **agent** chooses between $k$ **actions** and receives a **reward** based on the action it chooses. (⇒ **decision making under uncertainty**)
- **Action-Values**:
  - **Value**: expected reward

$$q_*(a) \coloneqq E[R_t | A_t = a] \quad \forall a \in \{1, \dots, k\}$$
$$= \sum_r p(r|a) \cdot r$$

  - <u>Goal</u>: maximize the expected reward

$$\underset{a}{\operatorname{argmax}}\, q_*(a)$$

  - <u>Estimation</u>: **Sample-average method**

$$Q_t(a) \coloneqq \frac{\sum_{i=1}^{t-1} R_i}{t-1}$$
$$= \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\#\text{times } a \text{ taken prior to } t}$$

  - **Greedy action**

$$a_g = \underset{a}{\operatorname{argmax}}\, Q_t(a)$$

    - = the action with the largest estimated value among all actions

- **Non-stationary bandit problem**
  - = when the reward distribution from certain action changes with time
  - **Incremental update rule**

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

$$\text{New Estimate} = \text{Old estimate} + \text{Stepsize} * (\text{Target} - \text{Old estimate})$$

- $a_n \in [0,1]$ (Sample-average: $a_n = \frac{1}{n}$)
- Derivation:

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

- Constant $a_n \rightarrow$ decaying past rewards

$$
\begin{aligned}
Q_{n+1} &= Q_n + a(R_n - Q_n) \\
&= aR_n + (1-a)Q_n \\
&= aR_n + (1-a)(aR_{n-1} + (1-a)Q_{n-1}) \\
&= \cdots \\
&= (1-a)^n Q_1 + \sum_{i=1}^{n} a(1-a)^{n-1} R_i
\end{aligned}
$$

$\Rightarrow$ Contribution of $Q_1$ decreases exponentially with time

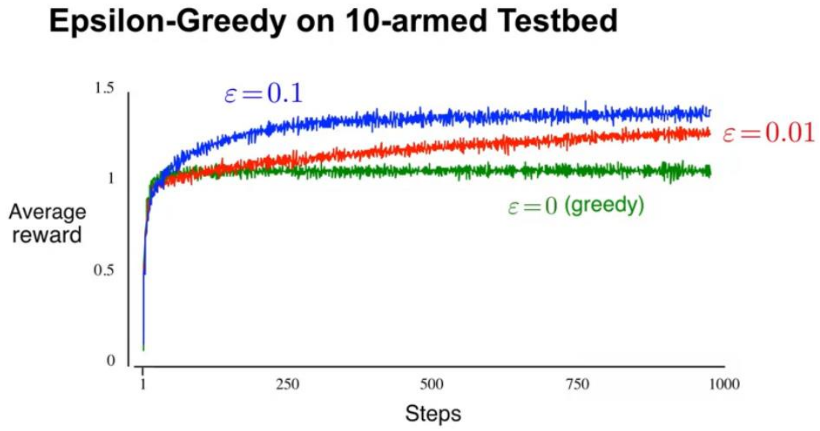
$\Rightarrow$ Rewards further back in time contribute exponentially less to sum

$\Rightarrow$ Most recent rewards contribute most to the current estimation

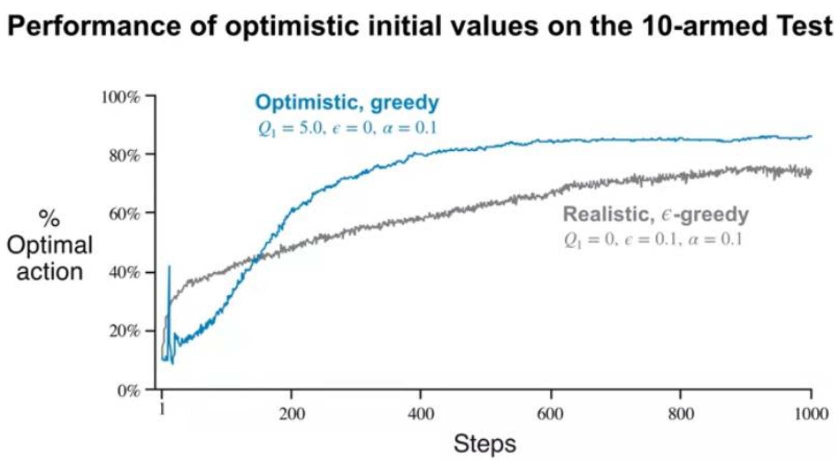
- **Exploration-exploitation dilemma**
  - **Exploration**: **explore** knowledge for **long-term** benefits (non $a_g$)
  - **Exploitation**: **exploit** knowledge for **short-term** benefits ($a_g$)
  - **Dilemma**: No agent can choose both exploitation & exploration at the same time.
  - **Selection 1: Epsilon-Greedy Action Selection**

$$
A_t \leftarrow \begin{cases} a_g = \underset{a}{\operatorname{argmax}}\, Q_t(a) & p = 1-\epsilon \\ a \sim \text{Uniform}(\{a_1, \dots, a_k\}) & p = \epsilon \end{cases}
$$

- $\epsilon$: probability of choosing to explore
- Impact of different $\epsilon$:




- **Optimistic initial values**
  - Procedure:
    - ⇒ Initialize $Q_1(a)$ with **reasonably large values** so that the first time $a$ is chosen, the observed reward will most likely be smaller than the optimistic initial estimate
    - ⇒ other actions will always look more appealing than $a$
    - ⇒ encourage exploration at the beginning
  - Performance:



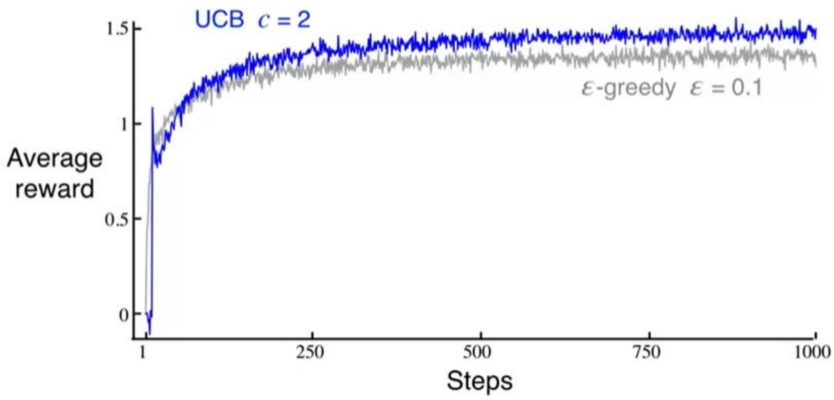
- ♦ <u>Limitations</u>:
  - ⇒ The only benefit is to drive **early exploration**
  - ⇒ Not well-suited for **non-stationary problems**
  - ⇒ Hard to decide the **best** optimistic initial values

- o **Selection 2: Upper-Confidence Bound [UCB] Action Selection**

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

- ♦ = select the action with the highest UCB (⇒ uncertainties in estimate)
- ♦ $Q_t(a)$: exploitation part
- ♦ $c\sqrt{\frac{\ln t}{N_t(a)}}$: exploration part
  - ⇒ $c$: user-specified param that controls #exploration
  - ⇒ $t$: timesteps
  - ⇒ $N_t(a)$: #times action $a$ was taken
- ♦ <u>Performance</u>:

**Performance of optimistic initial values on the 10-armed Testbed**



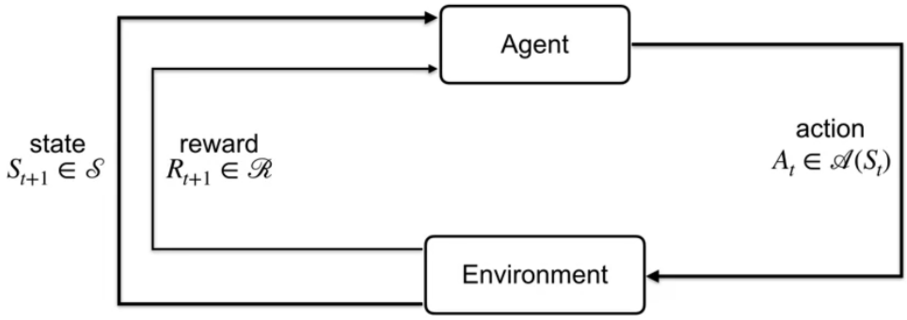

- **Contextual Bandits** (Real-World RL)
  - o <u>Problem</u>: Simulator ≠ Reality
  - o Shift the **priorities**:

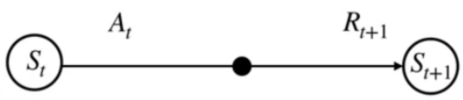
- ~~Temporal credit assignment~~ Generalization
- ~~Control environment~~ Environment control
- ~~Computational efficiency~~ Statistical efficiency
- ~~State~~ Features
- ~~Learning~~ Evaluation
- ~~Last policy~~ Every policy

- Contextual Bandits
  - Repeat:
    - $\Rightarrow$ Observe features $x$
    - $\Rightarrow$ Choose action $a \in A$
    - $\Rightarrow$ Observe reward $r$
  - Goal: maximize reward

# Markov Decision Processes

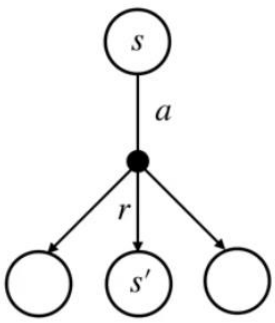
- <u>Problem</u>: Different states call for different actions
- <u>Framework</u>: discrete timesteps




  - Environment → Agent: state info $S_t$ (from set of states $\mathcal{S}$)
  - Agent → Environment: action in response $A_t$ (from set of actions $\mathcal{A}(S_t)$)
  - Environment → Agent: new state $S_{t+1}$ based on the action
  - Environment → Agent: reward $R_{t+1}$ based on the action
  - …… and it cycles ……

- <u>Dynamics of MDP</u>:

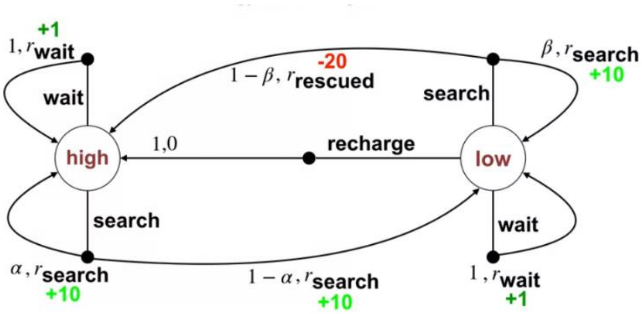



Transition function: $p(s', r|s, a)$

  - <u>Condition</u>: $\mathcal{S}$ & $\mathcal{A}$ are **finite sets**
  - <u>Properties</u>:
    - ♦ $p: \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0,1]$
    - ♦ $\sum_{s'} \sum_r p(s', r|s, a) = 1 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$

- **Markov Property**: The present state contains all the info necessary to predict the future.
  - ⇒ The present state is **efficient**.
  - ⇒ The info memorized from previous states don't matter.
- **MDP Formalism is abstract & flexible**:
  - e.g. recycling robot
    - Battery: $\mathcal{S} = \{\text{low}, \text{high}\}$
    - Action: $\mathcal{A}(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$

      $\mathcal{A}(\text{high}) = \{\text{search}, \text{wait}\}$
  - Transition dynamics:




    - Situation 1: wait

      ⇒ No prob involved.

      ⇒ $r_{\text{wait}} = +1$
    - Situation 2: recharge

      ⇒ No prob involved.

      ⇒ $r_{\text{recharge}} = 0$
    - Situation 3: search with "high" may reduce battery to "low"

      ⇒ Assume $\alpha$ = prob that battery stays "high".

      ⇒ In both $\alpha$ & $1 - \alpha$, the action yields a reward of $r_{\text{search}} = +10$.
    - Situation 4: search with "low" may deplete the battery

      ⇒ Assume $\beta$ = prob that battery isn't depleted.

      ⇒ If $1 - \beta$, it needs to be rescued ($r_{\text{rescued}} = -20$).

      ⇒ If $\beta$, the battery stays alive, then $r_{\text{search}} = +10$

# Goal of RL

- **Reward Hypothesis**: Agents should maximize the expected total rewards

$$E[G_t] = E[R_{t+1} + R_{t+2} + \cdots + R_T]$$

  - ⇒ maximizing immediate rewards doesn't always maximize the total rewards. In fact, it might lead to total failure.

- **Episodic Tasks**
  - Interaction breaks into **episodes**.
  - Each episode ends in a **terminal state**, at which the agent refreshes.
  - Episodes are **independent**.
  - Total reward:

$$G_t \coloneqq \sum_{k=1}^{T-t} R_{t+k}$$

- **Continuing Tasks**
  - Interactions goes on **continually**.
  - No terminal state.
  - Total reward:

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

    - $\gamma \in [0,1)$: discount factor (discount the rewards in the future)
    - Upper bound: $G_t \leq \frac{R_{\max}}{1-\gamma}$ (geometric series)
    - $\gamma = 0$: $G_t = R_{t+1}$ ⇒ **short-sighted** agent
    - $\gamma \to 1$: $G_t \to \sum_{k=1}^{\infty} R_{t+k}$ ⇒ **far-sighted** agent

  - **Recursive nature** of returns

$$G_t = R_{t+1} + \gamma G_{t+1}$$

# Policies & Optimality

- **Policy**: state → prob distribution for an action
  - **Deterministic Policy**: maps each state to an action

$$\pi(s) = a$$

  - **Stochastic Policy**: maps each state to multiple possible actions

$$\pi(a|s) \geq 0, \sum_{a \in \mathcal{A}(s)} \pi(a|s) = 1$$

  - **Valid vs Invalid Policies**: current action should be chosen depending **ONLY on the info of the current state**.
    - e.g. alternating between 2 actions is invalid
    - e.g. giving each of the 2 actions a 50% probability is valid

- **Value Funcs**: predict rewards into the future
  - **State-value func**: expected return from a given state under a specific policy

$$v_\pi(s) \coloneqq E_\pi[G_t | S_t = s]$$

  - **Action-value func**: expected return from a given state after taking a specific action, later following a specific policy

$$q_\pi(s, a) \coloneqq E_\pi[G_t | S_t = s, A_t = a]$$

  - <u>Benefits</u>:
    - Return is not immediately available.
    - Return may be random ← stochasticity in policy & env dynamics.

- **Bellman Equation**: relate current state to future states without waiting to observe future rewards
  - **State-value Bellman**:

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

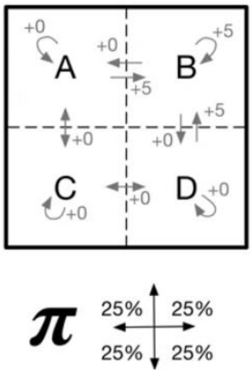
- Explanation:
  - ⇒ Step 1: Recursive nature of returns: $G_t \rightarrow R_{t+1} + \gamma G_{t+1}$
  - ⇒ Step 2:
    - Expand the expected return over all possible actions at the current state $s$. (⇐ current action ONLY depends on current state)
    - Expand over all possible rewards on all the possible next states based on current state $s$ and current action $a$. (⇐ next state & reward ONLY depend on the current state & action)
    - Expected return ⇒ weighted sum of all possible returns
  - ⇒ Step 3: Valid policy $\pi$ ONLY depends on current state's info (~~time~~)

- **Action-value Bellman**:

$$
\begin{aligned}
q_\pi(s, a) &\coloneqq E_\pi[G_t|S_t = s, A_t = a] \\
&= \sum_{s'} \sum_r p(s', r|s, a)(r + \gamma E_\pi[G_{t+1}|S_{t+1} = s']) \\
&= \sum_{s'} \sum_r p(s', r|s, a)\left(r + \gamma \sum_{a'} \pi(a'|s') E_\pi[G_{t+1}|S_{t+1} = s', A_{t+1} = a']\right) \\
&= \sum_{s'} \sum_r p(s', r|s, a)\left(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')\right)
\end{aligned}
$$

- Explanation:
  - ⇒ Step 1: State-value Bellman equation (action is already fixed)
  - ⇒ Step 2: Expand over all possible actions at the next state $s'$
  - ⇒ Step 3: Valid policy $\pi$ ONLY depends on current state's info (~~time~~)

- Example: Gridworld




- We have a 2x2 grid. Only landing in cell B gives a reward of +5.
- Our policy is a uniform 25% chance of moving in one of the 4 directions.
- Assume $\gamma = 0.7$. Calculate the value of each state:

$$\because v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\big(r + \gamma v_\pi(s')\big)$$

$$\therefore V_\pi(A) = \sum_a \pi(a|A)\big(r + 0.7V_\pi(s')\big)$$

$$= \frac{1}{4}\big(5 + 0.7V_\pi(B)\big) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(A)$$

$$V_\pi(B) = \frac{1}{4}0.7V_\pi(A) + \frac{1}{4}0.7V_\pi(D) + \frac{1}{2}\big(5 + 0.7V_\pi(B)\big)$$

$$V_\pi(C) = \frac{1}{4}0.7V_\pi(A) + \frac{1}{4}0.7V_\pi(D) + \frac{1}{2}0.7V_\pi(C)$$

$$V_\pi(D) = \frac{1}{4}\big(5 + 0.7V_\pi(B)\big) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(D)$$

* $p(s', r|s, a) = 1\ \forall a$ since the probability of landing to the next cell for all actions is absolute.

- Solve the 4 linear equations to get the value of each state:

$$V_\pi(A) = V_\pi(D) = 4.2, V_\pi(C) = 2.2, V_\pi(B) = 6.1$$

- Lesson: we can only apply this **DIRECTLY to small MDPs**. You do not want to solve over $10^{45}$ linear equations for the value function for Chess.

- **Optimal policy**
  - **Better**: $\pi_1 \geq \pi_2$ iff $\forall s \in \mathcal{S}: v_{\pi_1}(s) \geq v_{\pi_2}(s)$
  - **Optimal**: $\pi_* \geq \pi_i \quad \forall \pi_i \in \Pi$

$$v_* \coloneqq v_{\pi_*}(s) = \max_{\pi} v_\pi(s) \quad \forall s \in \mathcal{S}$$
$$q_* \coloneqq q_{\pi_*}(s,a) = \max_{\pi} q_\pi(s,a) \quad \forall s \in \mathcal{S} \; \& \; a \in \mathcal{A}$$

  - <u>Existence</u>: $\exists \pi_* \in \Pi$
    - Assume two policies $\pi_1$ & $\pi_2$.
    - Assume $\pi_1 \geq \pi_2$ for the former states, while $\pi_1 \leq \pi_2$ for the latter states.
    - If we choose our policy $\pi_3 = \pi_1$ for the former and $\pi_3 = \pi_2$ for the latter,
    - Then $\pi_3 \geq \pi_1$ & $\pi_3 \geq \pi_2$ always holds for all states.
    - Similarly, there is always an optimal policy for all states.

  - **Brute-Force search**: compute $v_{\pi_i}(s)\ \forall \pi_i$ to find the optimal policy
    - We can only apply Brute-Force search **DIRECTLY to small MDPs**.

      A general MDP contains $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies.

  - **Bellman Optimality Equation**
    - **State**:

$$v_*(s) = \sum_a \pi_*(a|s) \sum_{s'} \sum_r p(s',r|s,a)\big(r + \gamma v_*(s')\big)$$
$$= \max_a \sum_{s'} \sum_r p(s',r|s,a)\big(r + \gamma v_*(s')\big)$$

    - **Action**:

$$q_*(s) = \sum_{s'} \sum_r p(s',r|s,a)\left(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')\right)$$
$$= \sum_{s'} \sum_r p(s',r|s,a)\left(r + \gamma \max_{a'} q_*(s',a')\right)$$

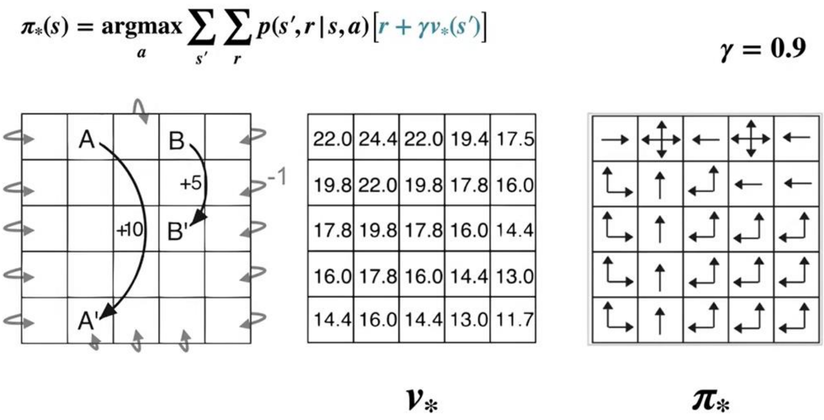
- Problem:
  - $\Rightarrow$ We can solve Bellman Equations with linear algebra. However,
  - $\Rightarrow$ We cannot solve Bellman Optimality Equations with linear algebra:

$$\pi, p, \gamma \rightarrow \text{Linear Algebra} \rightarrow v_\pi$$
$$\cancel{\pi_*}, p, \gamma \rightarrow \cancel{\text{Linear Algebra}} \rightarrow v_*$$

  - $\Rightarrow$ Why?
    - $\max \rightarrow$ nonlinearity
    - We do NOT know the optimal policy $\pi_*$. (Otherwise we already achieved the goal of RL)

- Optimal value funcs → Optimal policies

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




| | | | | |
|---|---|---|---|---|
| 22.0 | 24.4 | 22.0 | 19.4 | 17.5 |
| 19.8 | 22.0 | 19.8 | 17.8 | 16.0 |
| 17.8 | 19.8 | 17.8 | 16.0 | 14.4 |
| 16.0 | 17.8 | 16.0 | 14.4 | 13.0 |
| 14.4 | 16.0 | 14.4 | 13.0 | 11.7 |

- Problem → $v_*$ → $\pi_*$
- e.g. current state $s$ = upper right corner
  - $\Rightarrow$ $a$ = moving left: $v(s) = 0 + 0.9 \times 19.4 = 17.46$
  - $\Rightarrow$ $a$ = moving down: $v(s) = 0 + 0.9 \times 16.0 = 14.4 < 17.46$
  - $\Rightarrow$ $\therefore v_*(s) = 17.46$, $\pi_*(s)$ = moving left
- e.g. current state $s$ = A
  - $\Rightarrow$ $a$ has to be jumping to A′: $v(s) = 10 + 0.9 * 16.0 = 24.4$

$\Rightarrow\ \therefore v_*(s) = 24.4,\ \ \pi_*(s)$ = any action since it will always jump to A′

- e.g. current state $s$ = middle of right edge

  $\Rightarrow\ a$ = moving left or up: $v(s) = 0 + 0.9 \times 16.0 = 14.4$

  $\Rightarrow\ a$ = moving down: $v(s) = 0 + 0.9 \times 13.0 = 11.7$

  $\Rightarrow\ a$ = moving right: $v(s) = -1 + 0.9 \times 14.4 = 11.96$

  $\Rightarrow\ \therefore v_*(s) = 14.4,\ \ \pi_*(s)$ = moving left or up

- Similarly for all cells, we can find $\pi_*$ from $v_*$.

# Dynamic Programming

- **Policy Evaluation** (i.e. **Prediction**)
  - = the task of determining the **value func** for a specific policy.
    - In theory: $\pi, p, \gamma \to \text{LinAlg} \to v_\pi$
    - In practice: $\pi, p, \gamma \to \text{DP} \to v_\pi$
    - $p$: probability = dynamics of the environment
  - **Iterative Policy Evaluation**
    - Update rule:

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\big(r + \gamma v_k(s')\big)$$

    - Goal: approximate to the optimal $v$ (when $v_{k+1} = v_k$)
    - Algorithm:

Given $\pi, \epsilon$. Init $V = \mathbf{0}, V' = \mathbf{0}, \Delta \geq \epsilon$
While $\Delta \geq \epsilon$:
  $\Delta \leftarrow 0$
  For $s$ in $\mathcal{S}$:
    $V'(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\big(r + \gamma V(s')\big)$
    $\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$
  $V \leftarrow V'$
$v_\pi \leftarrow V$

- **Policy Iteration** (i.e. **Control**)
  - = the task of finding a **policy** that maximizes the value func.
    - In practice: $p, \gamma \to \text{DP} \to \pi_*$
  - **Policy Improvement Theorem**

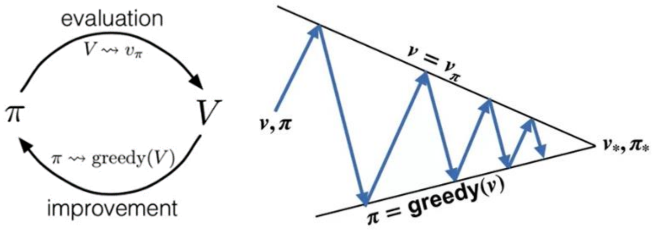
$$\forall s \in \mathcal{S}: q_\pi\big(s, \pi'(s)\big) \geq q_\pi\big(s, \pi(s)\big) \Rightarrow \pi' \geq \pi$$
$$\exists s \in \mathcal{S}: q_\pi\big(s, \pi'(s)\big) > q_\pi\big(s, \pi(s)\big) \Rightarrow \pi' > \pi$$

- **Greedified policy** (always choose greedy action):

$$\pi'(s) \coloneqq \underset{a}{\operatorname{argmax}} \sum_{s'} \sum_{r} p(s', r|s, a)\big(r + \gamma v_\pi(s')\big)$$

- Interpretation: $\pi'$ is a strict improvement unless $\pi$ was already optimal.

- **Policy Iteration = Evaluation & Improvement**




$$\pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} v_{\pi_2} \xrightarrow{\text{I}} \pi_3 \xrightarrow{\text{E}} \ldots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_{\pi_*} \xrightarrow{\text{I}} \pi_*$$

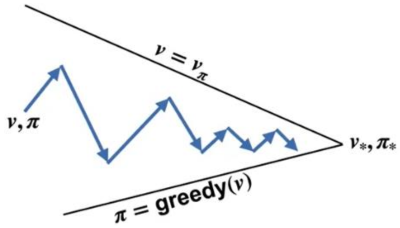
- All policies in the iteration are deterministic → $\exists \pi_*$
- Algorithm:

```
# Initialization
Init ϵ, Δ ≥ ϵ
For s in S:
    Init π(s) ∈ A(s), V(s) ∈ ℝ

# Policy Evaluation
While Δ ≥ ϵ:
    Δ ← 0
    For s in S:
        v ← V(s)
        V(s) ← Σ_{s'} Σ_r p(s', r|s, π(s))(r + γV(s'))
        Δ ← max(Δ, |v − V(s)|)

# Policy Improvement
Policy_is_stable ← true
For s in S:
    Old_action ← π(s)
    π(s) ← argmax_a Σ_{s'} Σ_r p(s', r|s, a)(r + γv_π(s'))
    If Old_action ≠ π(s):
        Policy_is_stable ← false
If Policy_is_stable:
    return V, π
Else:
    # Policy Evaluation
```

- **Generalized Policy Iteration**




  - **Value Iteration**

| |
|---|
| *# Initialization*<br>Init $\epsilon, \Delta \geq \epsilon$<br>For $s$ in $\mathcal{S}$:<br>  Init $V(s) \in \mathbb{R}$<br><br>*# Policy Evaluation*<br>While $\Delta \geq \epsilon$:<br>  $\Delta \leftarrow 0$<br>  For $s$ in $\mathcal{S}$:<br>    $v \leftarrow V(s)$<br>    $V(s) \leftarrow \max_a \sum_{s',r} p(s',r\|s,a)\big(r + \gamma V(s')\big)$<br>    $\Delta \leftarrow \max(\Delta, \|v - V(s)\|)$<br><br>*# Output*<br>Return $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r\|s,a)\big(r + \gamma V(s')\big)$ |

    - = a combination of PE & PI into a single update
    - **Synchronous**: sweep the entire state space on each iteration (If state space too large → time-consuming)
    - **Asynchronous**: sweep whichever states on each iteration

  - **Efficiency of Dynamic Programming**

    - <u>PE sampling alternative</u>:

      ⇒ **Monte-Carlo Method**: estimates each state value **independently**. (However, there is too much randomness in the sampling process, making it hard for the estimation to converge)

$$v_\pi(s) \coloneqq E_\pi[G_t | S_t = S]$$

⇒ **Bootstrapping**: estimates each state value **based on successor states**. (much more efficient than Monte-Carlo)

$$v_{k+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a) \left(r + \gamma v_k(s')\right)$$

- PI alternative:

  ⇒ **Brute-Force Search**: evaluates every possible deterministic policy one at a time. (However, #policies = $|\mathcal{A}|^{|\mathcal{S}|}$, inefficient)

  ⇒ **Policy Improvement**: $\pi_1 \to \pi_2 \to \cdots \to \pi_*$

  (Polynomial instead of exponential)

- **Curse of Dimensionality**: the size of $\mathcal{S}$ grows exponentially as the number of relevant features increases.