Introduction to Reinforcement Learning

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Focus Programme in Formal Methods and Artificial Intelligenc
IRL ReLaX
6 February 2026

Supervised Learning - Classification / Prediction from training data

Unsupervised Learning - "Extract" information

Clustering

Reinforcement Learning - Sequence of steps

Playing a game

Repeatedly play the game

Learn a good strategy from outcome

Andrew Barto & Richard Sutton

Turing Award 2024

Book, 2nd ed 2018

An alternative approach to learning

- Supervised learning — use labelled examples to learn a classifier

An alternative approach to learning

- Supervised learning — use labelled examples to learn a classifier
- Unsupervised learning — search for patterns, structure in data

An alternative approach to learning

- Supervised learning — use labelled examples to learn a classifier
- Unsupervised learning — search for patterns, structure in data
- Reinforcement learning — learning through interaction
  - Choose actions in an uncertain environment
  - Actions change state, yield rewards
  - Learn optimal strategies to maximize long term rewards

An alternative approach to learning

- Supervised learning — use labelled examples to learn a classifier
- Unsupervised learning — search for patterns, structure in data
- Reinforcement learning — learning through interaction
  - Choose actions in an uncertain environment
  - Actions change state, yield rewards
  - Learn optimal strategies to maximize long term rewards
- Examples
  - Playing games — AlphaGo, reward is result of the game
  - Motion planning — robot searching for an optimal path with obstacles
  - Feedback control — balancing an object

The components

- Policy What action to take in the current state
  - "Strategy", can be probabilistic

The components

- Policy What action to take in the current state
  - "Strategy", can be probabilistic
- Reward In response to taking an action
  - Short-term outcome, may be negative or positive

The components

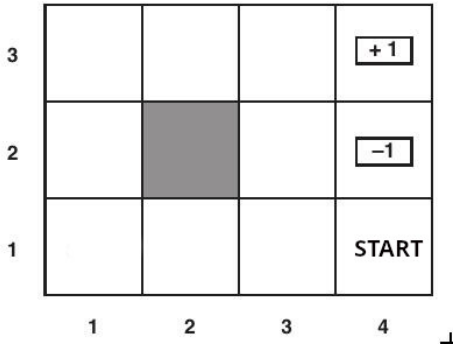
- Policy What action to take in the current state
  - "Strategy", can be probabilistic
- Reward In response to taking an action
  - Short-term outcome, may be negative or positive
- Value Accumulation of rewards over future actions
  - Long-term outcome, goal is to maximize value

The components

- Policy What action to take in the current state
  - "Strategy", can be probabilistic
- Reward In response to taking an action
  - Short-term outcome, may be negative or positive
- Value Accumulation of rewards over future actions
  - Long-term outcome, goal is to maximize value
- Environment Model How the environment will behave
  - Given a state and action, what is the next state, reward?
  - Probabilistic, in general
  - Use models for *planning*
  - Can also use RL without models, trial-and-error learners

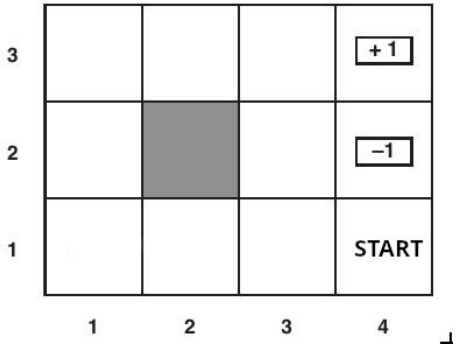
Motion planning example

- 4×3 grid
- Rewards are attached to states
  - Two terminal states with rewards $+1$, -1
  - All other states have reward -0.04
  - Move till you reach a terminal state
  - Maximize the sum of the rewards seen



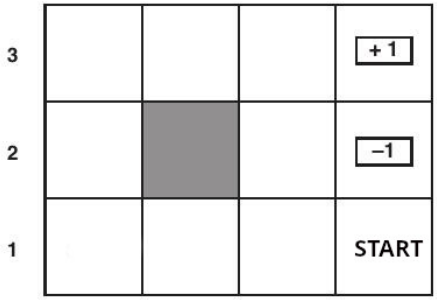
Motion planning example

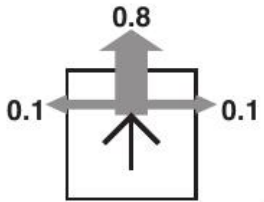
- 4×3 grid
- Rewards are attached to states
  - Two terminal states with rewards $+1$, -1
  - All other states have reward -0.04
  - Move till you reach a terminal state
  - Maximize the sum of the rewards seen
- Policy — which direction to move from a given square in the grid



Motion planning example

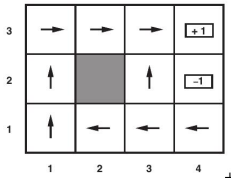
- 4×3 grid
- Rewards are attached to states
  - Two terminal states with rewards $+1$, -1
  - All other states have reward -0.04
  - Move till you reach a terminal state
  - Maximize the sum of the rewards seen
- Policy — which direction to move from a given square in the grid
- Outcome of action is nondeterministic
  - With probability 0.8, go in intended direction
  - With probability 0.2, deflect at right angles
  - Collision with boundary keeps you stationary





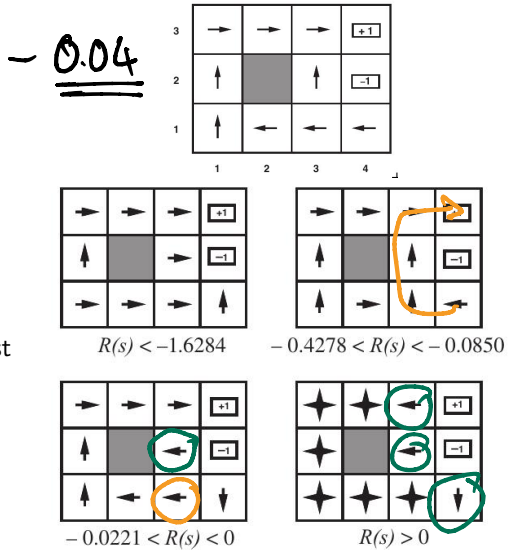
Motion planning example

- Optimal policy learned by repeatedly moving on the board
  - From bottom right, conservatively follow the long route around the obstacle to avoid -1



Motion planning example

- Optimal policy learned by repeatedly moving on the board
  - From bottom right, conservatively follow the long route around the obstacle to avoid -1
- Optimal policies for different value of $R(s)$, reward for non-final states
  - If $R(s) < -1.6284$, terminate as fast as possible
  - If $-0.4278 < R(s) < -0.0850$, risk going past -1 to reach $+1$ quickly
  - If $-0.0221 < R(s) < 0$, take no risks, avoid -1 at all cost
  - If $R(s) > 0$ avoid terminating



Exploration vs exploitation

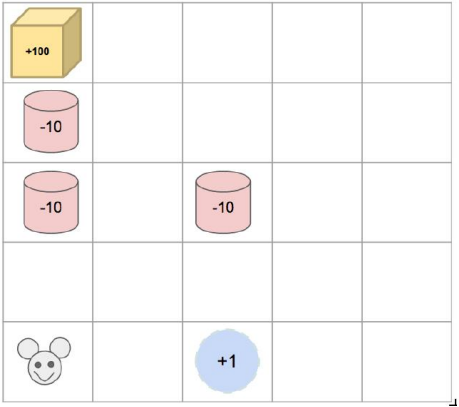
- Policy evolves by experience

Exploration vs exploitation

- Policy evolves by experience
- Greedy strategy is to always choose best known option

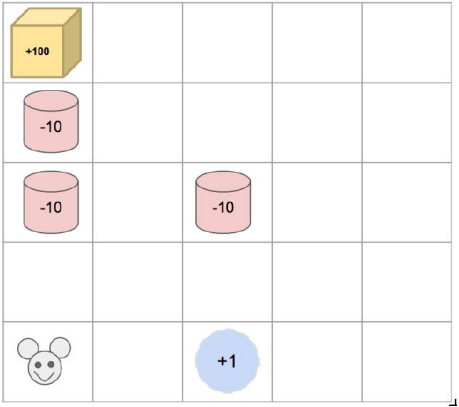
Exploration vs exploitation

- Policy evolves by experience
- Greedy strategy is to always choose best known option
- Using this we may get stuck in a local optimum
  - Greedy strategy only allows the mouse to discover water with reward $+1$
  - Mouse never discovers a path to cheese with $+100$ because of negative rewards en route



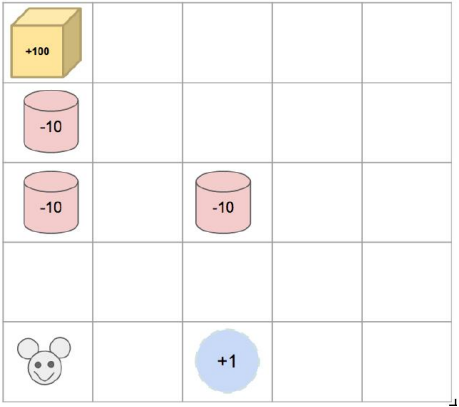
Exploration vs exploitation

- Policy evolves by experience
- Greedy strategy is to always choose best known option
- Using this we may get stuck in a local optimum
  - Greedy strategy only allows the mouse to discover water with reward $+1$
  - Mouse never discovers a path to cheese with $+100$ because of negative rewards en route
- How to balance exploitation (greedy) vs exploration?



Exploration vs exploitation

- Policy evolves by experience
- Greedy strategy is to always choose best known option
- Using this we may get stuck in a local optimum
  - Greedy strategy only allows the mouse to discover water with reward $+1$
  - Mouse never discovers a path to cheese with $+100$ because of negative rewards en route
- How to balance exploitation (greedy) vs exploration?
- Formalize these ideas using Markov Decision Processes



Bandits

- One-armed bandit — slang for a slot machine in a casino
  - Put in a coin and pull a lever (the arm)
  - With high probability, lose your coin (the bandit steals your money)
  - With low probability, get varying reward, rewards follow some probability distribution

Bandits

- One-armed bandit — slang for a slot machine in a casino
  - Put in a coin and pull a lever (the arm)
  - With high probability, lose your coin (the bandit steals your money)
  - With low probability, get varying reward, rewards follow some probability distribution
- k-armed bandit
  - Each arm has a different reward probability
  - Goal is to maximize total reward over a sequence of plays

Bandits

- One-armed bandit — slang for a slot machine in a casino
  - Put in a coin and pull a lever (the arm)
  - With high probability, lose your coin (the bandit steals your money)
  - With low probability, get varying reward, rewards follow some probability distribution
- k-armed bandit
  - Each arm has a different reward probability
  - Goal is to maximize total reward over a sequence of plays
- Action corresponds to choosing the arm

Bandits

- One-armed bandit — slang for a slot machine in a casino
  - Put in a coin and pull a lever (the arm)
  - With high probability, lose your coin (the bandit steals your money)
  - With low probability, get varying reward, rewards follow some probability distribution
- k-armed bandit
  - Each arm has a different reward probability
  - Goal is to maximize total reward over a sequence of plays
- Action corresponds to choosing the arm
  - For each action a, $q_*(a)$ is expected reward if we choose a

Bandits

- One-armed bandit — slang for a slot machine in a casino
  - Put in a coin and pull a lever (the arm)
  - With high probability, lose your coin (the bandit steals your money)
  - With low probability, get varying reward, rewards follow some probability distribution
- k-armed bandit
  - Each arm has a different reward probability
  - Goal is to maximize total reward over a sequence of plays
- Action corresponds to choosing the arm
  - For each action a, $q_*(a)$ is expected reward if we choose a
  - A_t is action chosen at time t, with reward R_t

Bandits

- One-armed bandit — slang for a slot machine in a casino
  - Put in a coin and pull a lever (the arm)
  - With high probability, lose your coin (the bandit steals your money)
  - With low probability, get varying reward, rewards follow some probability distribution
- k-armed bandit
  - Each arm has a different reward probability
  - Goal is to maximize total reward over a sequence of plays
- Action corresponds to choosing the arm
  - For each action a, $q_*(a)$ is expected reward if we choose a
  - A_t is action chosen at time t, with reward R_t
  - If we knew $q_*(a)$ we would always choose $A_t = \arg\max_a q_*(a)$

Bandits

- One-armed bandit — slang for a slot machine in a casino
  - Put in a coin and pull a lever (the arm)
  - With high probability, lose your coin (the bandit steals your money)
  - With low probability, get varying reward, rewards follow some probability distribution
- k-armed bandit
  - Each arm has a different reward probability
  - Goal is to maximize total reward over a sequence of plays
- Action corresponds to choosing the arm
  - For each action a, $q_*(a)$ is expected reward if we choose a
  - A_t is action chosen at time t, with reward R_t
  - If we knew $q_*(a)$ we would always choose $A_t = \arg\max_a q_*(a)$
  - Assume $q_*(a)$ is unknown — build an estimate $Q_t(a)$ of $q_*(a)$ at time t

Exploration and exploitation

- Build $Q_t(a)$, estimate of $q_*(a)$ at time t, from past observations (sample average)

$$\frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$$

Exploration and exploitation

- Build $Q_t(a)$, estimate of $q_*(a)$ at time t, from past observations (sample average)

$$\frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$$

- Greedy policy chooses $\arg\max_a Q_t(a)$
- How will we learn about all actions?

Exploration and exploitation

- Build $Q_t(a)$, estimate of $q_*(a)$ at time t, from past observations (sample average)

$$\frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$$

- Greedy policy chooses $\arg\max_a Q_t(a)$
- How will we learn about all actions?
- ε-greedy policy
  - With small probability ε, choose a random action (uniform distribution)
  - With probability $1 - \varepsilon$, follow greedy

Exploration and exploitation

- Build $Q_t(a)$, estimate of $q_*(a)$ at time t, from past observations (sample average)

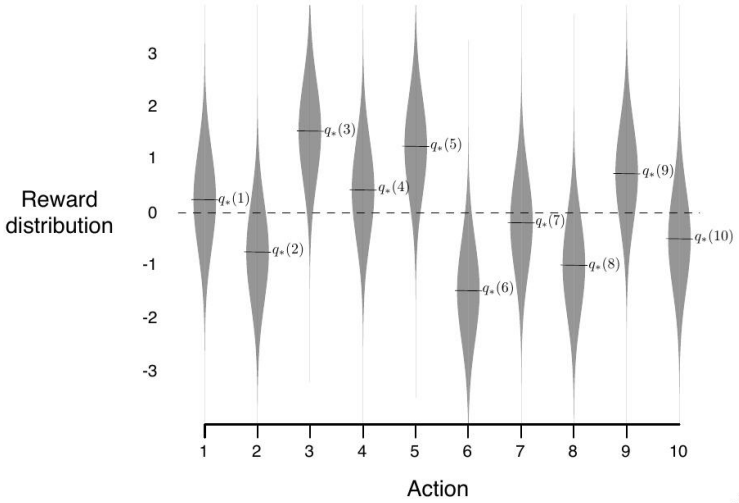
$$\frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$$

- Greedy policy chooses $\arg\max_a Q_t(a)$
- How will we learn about all actions?
- ε-greedy policy
  - With small probability ε, choose a random action (uniform distribution)
  - With probability $1 - \varepsilon$, follow greedy
- ε-greedy is a simple way to balance exploitation with exploration
  - Theoretically, explores all actions infinitely often
  - Practical effectiveness depends

Exploration and exploitation

10 bandit experiment

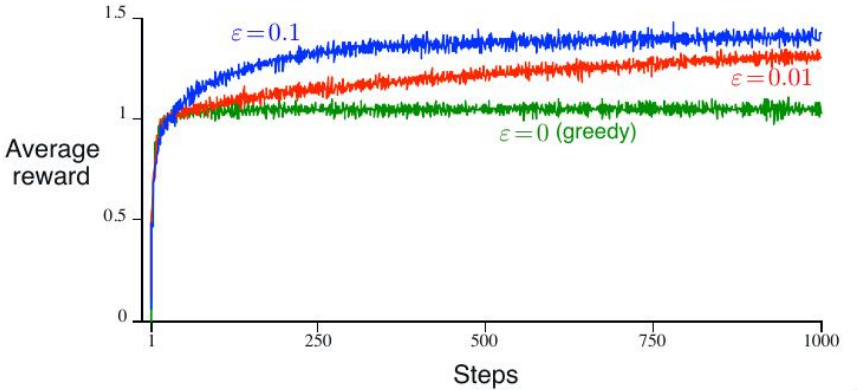
- Each bandit's reward follows Gaussian distribution
- Same variance, mean is chosen randomly



Exploration and exploitation

Performance of ε-greedy strategies

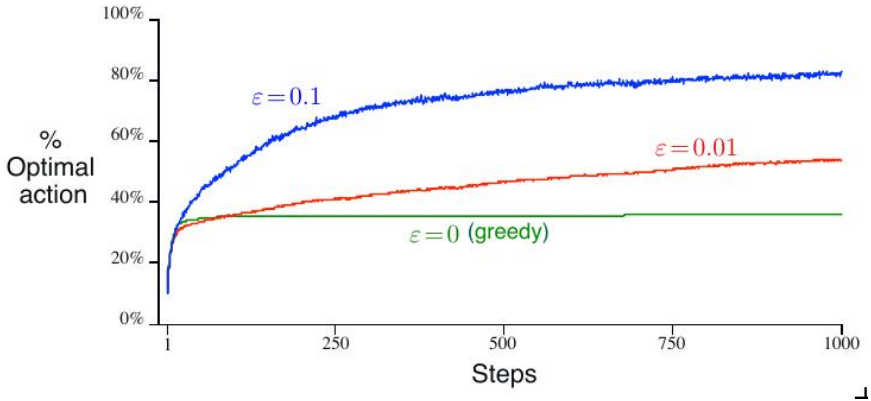
- Pure greedy strategy is sub-optimal
- Initial "learning rate" is more or less equal



Exploration and exploitation

Discovery of optimal actions

- Pure greedy strategy discovers optimal action only 1/3 of the time



Incremental calculation

- Focus on a single action a. Sample average is $\frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$

Incremental calculation

- Focus on a single action a. Sample average is $\frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$
- R_i — reward when a is selected for ith time
- Q_n — estimate of action value after a has been selected $n-1$ times

Incremental calculation

- Focus on a single action a. Sample average is $\frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$
- R_i — reward when a is selected for ith time
- Q_n — estimate of action value after a has been selected $n-1$ times
- $Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$

Incremental calculation

- Focus on a single action a. Sample average is $\frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$
- R_i — reward when a is selected for ith time
- Q_n — estimate of action value after a has been selected $n-1$ times
- $Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$
- $Q_{n+1} = \frac{1}{n} \sum_{i=1}^{n} R_i$

Incremental calculation

- Focus on a single action a. Sample average is $\frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$
- R_i — reward when a is selected for ith time
- Q_n — estimate of action value after a has been selected $n-1$ times
- $Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$
- $Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right)$

Incremental calculation

- Focus on a single action a. Sample average is $\frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$
- R_i — reward when a is selected for ith time
- Q_n — estimate of action value after a has been selected $n-1$ times
- $Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$
- $Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) = \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right)$

Incremental calculation

- Focus on a single action a. Sample average is $\frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$
- R_i — reward when a is selected for ith time
- Q_n — estimate of action value after a has been selected $n-1$ times
- $Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$
- $Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) = \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right)$
  $= \frac{1}{n}(R_n + (n-1)Q_n)$

Incremental calculation

- Focus on a single action a. Sample average is $\frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$
- R_i — reward when a is selected for ith time
- Q_n — estimate of action value after a has been selected $n-1$ times
- $Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$
- $Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) = \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right)$
  $= \frac{1}{n}(R_n + (n-1)Q_n) = \frac{1}{n}(R_n + nQ_n - Q_n)$

Incremental calculation

- Focus on a single action a. Sample average is $\frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$
- R_i — reward when a is selected for ith time
- Q_n — estimate of action value after a has been selected $n-1$ times
- $Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$
- $Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) = \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right)$
  $= \frac{1}{n}(R_n + (n-1)Q_n) = \frac{1}{n}(R_n + nQ_n - Q_n) = Q_n + \frac{1}{n}[R_n - Q_n]$

Incremental calculation

- Focus on a single action a. Sample average is $\frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_t=a}}$
- R_i — reward when a is selected for ith time
- Q_n — estimate of action value after a has been selected $n-1$ times
- $Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$
- $Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) = \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right)$
  $= \frac{1}{n}(R_n + (n-1)Q_n) = \frac{1}{n}(R_n + nQ_n - Q_n) = Q_n + \frac{1}{n}[R_n - Q_n]$
- We will see this pattern often:

  NewEstimate = OldEstimate + Step [Target - OldEstimate]

Stationary vs non-stationary

- Non-stationary: Reward probabilities change over time

Stationary vs non-stationary

- Non-stationary: Reward probabilities change over time
- Use a constant step $\alpha \in (0, 1]$ — $Q_{n+1} = Q_n + \alpha[R_n - Q_n]$

Stationary vs non-stationary

- Non-stationary: Reward probabilities change over time
- Use a constant step $\alpha \in (0, 1]$ — $Q_{n+1} = Q_n + \alpha[R_n - Q_n]$
- $Q_{n+1} = Q_n + \alpha[R_n - Q_n]$

Stationary vs non-stationary

- Non-stationary: Reward probabilities change over time
- Use a constant step $\alpha \in (0, 1]$ — $Q_{n+1} = Q_n + \alpha[R_n - Q_n]$
- $Q_{n+1} = Q_n + \alpha[R_n - Q_n] \ = \alpha R_n + (1 - \alpha)Q_n$

Stationary vs non-stationary

- Non-stationary: Reward probabilities change over time
- Use a constant step $\alpha \in (0, 1]$ — $Q_{n+1} = Q_n + \alpha[R_n - Q_n]$
- $Q_{n+1} = Q_n + \alpha[R_n - Q_n] = \alpha R_n + (1 - \alpha) Q_n$
  $= \alpha R_n + (1 - \alpha)[\alpha R_{n-1} + (1 - \alpha) Q_{n-1}]$

Stationary vs non-stationary

- Non-stationary: Reward probabilities change over time
- Use a constant step $\alpha \in (0, 1]$ — $Q_{n+1} = Q_n + \alpha[R_n - Q_n]$
- $$\begin{aligned} Q_{n+1} = Q_n + \alpha[R_n - Q_n] &= \alpha R_n + (1-\alpha)Q_n \\ &= \alpha R_n + (1-\alpha)[\alpha R_{n-1} + (1-\alpha)Q_{n-1}] \\ &= \alpha R_n + \alpha(1-\alpha)R_{n-1} + (1-\alpha)^2 Q_{n-1} \end{aligned}$$

Stationary vs non-stationary

- Non-stationary: Reward probabilities change over time
- Use a constant step $\alpha \in (0, 1]$ — $Q_{n+1} = Q_n + \alpha[R_n - Q_n]$
- $$\begin{aligned} Q_{n+1} &= Q_n + \alpha[R_n - Q_n] \quad = \alpha R_n + (1-\alpha)Q_n \\ &= \alpha R_n + (1-\alpha)[\alpha R_{n-1} + (1-\alpha)Q_{n-1}] \\ &= \alpha R_n + \alpha(1-\alpha)R_{n-1} + (1-\alpha)^2 Q_{n-1} \\ &= \alpha R_n + \alpha(1-\alpha)R_{n-1} + \alpha(1-\alpha)^2 R_{n-2} + \cdots + \alpha(1-\alpha)^{n-1} R_1 + (1-\alpha)^n Q_1 \end{aligned}$$

Stationary vs non-stationary

- Non-stationary: Reward probabilities change over time
- Use a constant step $\alpha \in (0, 1]$ — $Q_{n+1} = Q_n + \alpha[R_n - Q_n]$
- $$\begin{aligned} Q_{n+1} &= Q_n + \alpha[R_n - Q_n] \\ &= \alpha R_n + (1-\alpha)Q_n \\ &= \alpha R_n + (1-\alpha)[\alpha R_{n-1} + (1-\alpha)Q_{n-1}] \\ &= \alpha R_n + \alpha(1-\alpha)R_{n-1} + (1-\alpha)^2 Q_{n-1} \\ &= \alpha R_n + \alpha(1-\alpha)R_{n-1} + \alpha(1-\alpha)^2 R_{n-2} + \cdots + \alpha(1-\alpha)^{n-1} R_1 + (1-\alpha)^n Q_1 \end{aligned}$$

$$= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$$

Stationary vs non-stationary

- Non-stationary: Reward probabilities change over time
- Use a constant step $\alpha \in (0, 1]$ — $Q_{n+1} = Q_n + \alpha[R_n - Q_n]$
- $$\begin{aligned} Q_{n+1}^{a} &= Q_n + \alpha[R_n - Q_n] \quad = \alpha R_n + (1-\alpha)Q_n \\ &= \alpha R_n + (1-\alpha)[\alpha R_{n-1} + (1-\alpha)Q_{n-1}] \\ &= \alpha R_n + \alpha(1-\alpha)R_{n-1} + (1-\alpha)^2 Q_{n-1} \\ &= \alpha R_n + \alpha(1-\alpha)R_{n-1} + \alpha(1-\alpha)^2 R_{n-2} + \cdots + \alpha(1-\alpha)^{n-1} R_1 + (1-\alpha)^n Q_1 \end{aligned}$$

  $$= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$$

- Exponentially decaying weighted average of rewards

Stationary vs non-stationary

- Non-stationary: Reward probabilities change over time
- Use a constant step $\alpha \in (0, 1]$ — $Q_{n+1} = Q_n + \alpha[R_n - Q_n]$
- $$\begin{aligned} Q_{n+1} &= Q_n + \alpha[R_n - Q_n] \\ &= \alpha R_n + (1-\alpha)Q_n \\ &= \alpha R_n + (1-\alpha)[\alpha R_{n-1} + (1-\alpha)Q_{n-1}] \\ &= \alpha R_n + \alpha(1-\alpha)R_{n-1} + (1-\alpha)^2 Q_{n-1} \\ &= \alpha R_n + \alpha(1-\alpha)R_{n-1} + \alpha(1-\alpha)^2 R_{n-2} + \cdots + \alpha(1-\alpha)^{n-1}R_1 + (1-\alpha)^n Q_1 \\ &= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i \end{aligned}$$
- Exponentially decaying weighted average of rewards
- Initial value Q_1 affects the calculation — different heuristics possible

Summary

- k-armed bandit is the simplest interesting situation to analyze
- ε-greedy strategy balances exploration and exploitation
- Incremental update rule for estimates

  NewEstimate = OldEstimate + Step [Target - OldEstimate]
- Exponentially decaying weighted average when rewards change over time (non-stationary)

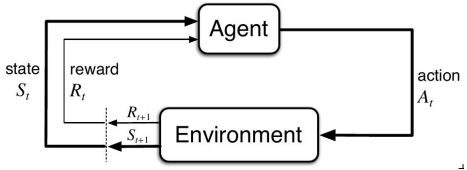
Markov Decision Processes

- Set of states S, actions A, rewards R

Markov Decision Processes

- Set of states S, actions A, rewards R
- At time t, agent in state S_t selects action A_t, moves to state S_{t+1} and receives reward R_{t+1}

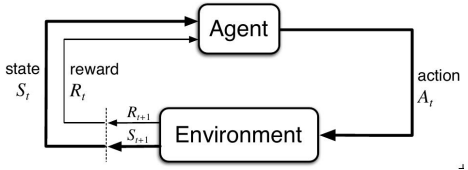
  Trajectory $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$

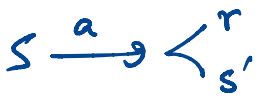


Markov Decision Processes

- Set of states S, actions A, rewards R
- At time t, agent in state S_t selects action A_t, moves to state S_{t+1} and receives reward R_{t+1}

  Trajectory $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$
- Probabilistic transition function: $p(s', r \mid s, a)$
  - Probability of moving to state s' with reward r if we choose a at s
  - For each (s, a), $\sum_{s'} \sum_{r} p(s', r \mid s, a) = 1$

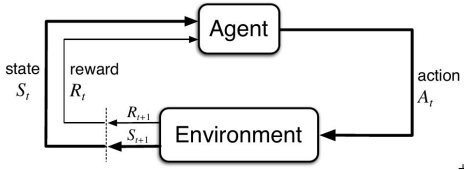


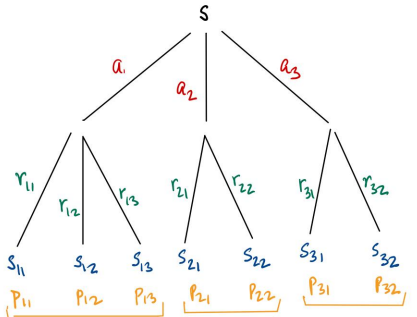


Markov Decision Processes

- Set of states S, actions A, rewards R
- At time t, agent in state S_t selects action A_t, moves to state S_{t+1} and receives reward R_{t+1}

  Trajectory $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$
- Probabilistic transition function: $p(s', r \mid s, a)$
  - Probability of moving to state s' with reward r if we choose a at s
  - For each (s, a), $\sum_{s'} \sum_r p(s', r \mid s, a) = 1$
  - Backup diagram

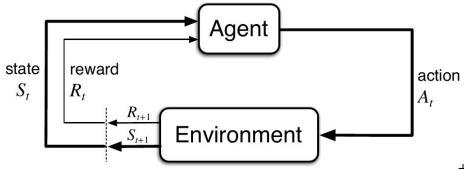


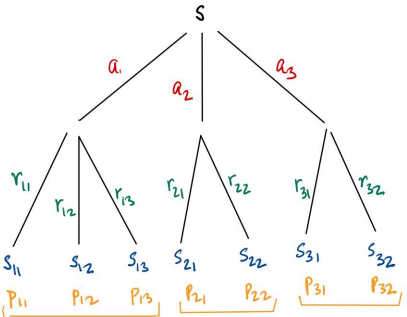


Markov Decision Processes

- Set of states S, actions A, rewards R
- At time t, agent in state S_t selects action A_t, moves to state S_{t+1} and receives reward R_{t+1}

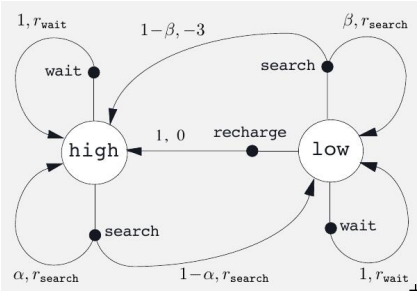
  Trajectory $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$
- Probabilistic transition function: $p(s', r \mid s, a)$
  - Probability of moving to state s' with reward r if we choose a at s
  - For each (s, a), $\sum_{s'} \sum_r p(s', r \mid s, a) = 1$
  - Backup diagram
- Typically assume finite MDPs — S, A and R are finite





MDP Example: Robot that collects empty cans

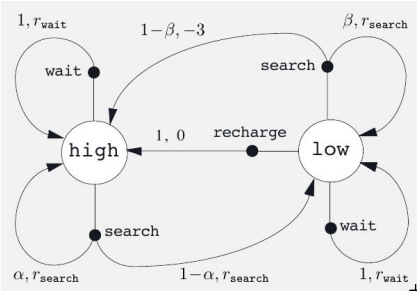
- State — battery charge: high, low
- Actions: search for a can, wait for someone to bring can, recharge battery
  - No recharge when high



s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1-\alpha$	r_{search}
low	search	high	$1-\beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-

MDP Example: Robot that collects empty cans

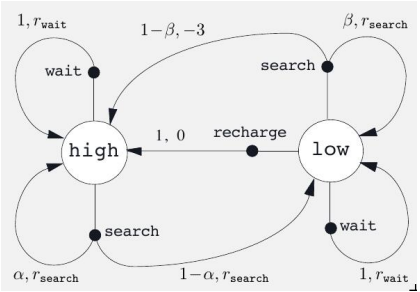
- State — battery charge: high, low
- Actions: search for a can, wait for someone to bring can, recharge battery
  - No recharge when high
- α, β, probabilities associated with change of battery state while searching



s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1-\alpha$	r_{search}
low	search	high	$1-\beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-

MDP Example: Robot that collects empty cans

- State — battery charge: high, low
- Actions: search for a can, wait for someone to bring can, recharge battery
  - No recharge when high
- α, β, probabilities associated with change of battery state while searching
- 1 unit of reward per can collected
- $r_{\text{search}} > r_{\text{wait}}$ — cans collected while searching, waiting
- Negative reward for requiring rescue (low to high while searching)



s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1-\alpha$	r_{search}
low	search	high	$1-\beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-

Long term rewards

- How do we formalize long term rewards?

Long term rewards

- How do we formalize long term rewards?
- Assume that each trajectory is a finite episode

Long term rewards

- How do we formalize long term rewards?
- Assume that each trajectory is a finite episode
- Episode with T steps, expected reward at time t: $G_t \triangleq R_{t+1} + R_{t+2} + \cdots + R_T$
  - Each episode is independent: rewards are reset after each episode

Long term rewards

- How do we formalize long term rewards?
- Assume that each trajectory is a finite episode
- Episode with T steps, expected reward at time t: $G_t \triangleq R_{t+1} + R_{t+2} + \cdots + R_T$
  - Each episode is independent: rewards are reset after each episode
- In some situations, trajectories may be (potentially) infinite
  - Discounted rewards: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, $0 \leq \gamma \leq 1$
- Inductive calculation of expected reward

$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$

Long term rewards

- How do we formalize long term rewards?
- Assume that each trajectory is a finite episode
- Episode with T steps, expected reward at time t: $G_t \triangleq R_{t+1} + R_{t+2} + \cdots + R_T$
  - Each episode is independent: rewards are reset after each episode
- In some situations, trajectories may be (potentially) infinite
  - Discounted rewards: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, $0 \leq \gamma \leq 1$
- Inductive calculation of expected reward

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

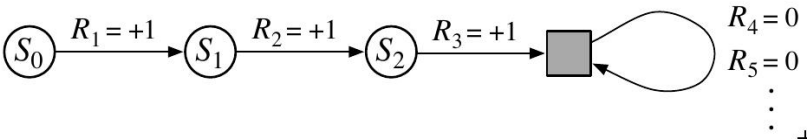
Long term rewards

- How do we formalize long term rewards?
- Assume that each trajectory is a finite episode
- Episode with T steps, expected reward at time t: $G_t \triangleq R_{t+1} + R_{t+2} + \cdots + R_T$
  - Each episode is independent: rewards are reset after each episode
- In some situations, trajectories may be (potentially) infinite
  - Discounted rewards: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, $0 \leq \gamma \leq 1$
- Inductive calculation of expected reward

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

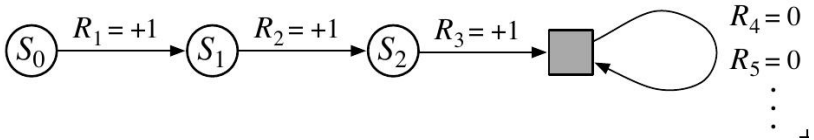
Long term rewards

- Can make all episodes infinite by adding a self-loop with reward 0



Long term rewards

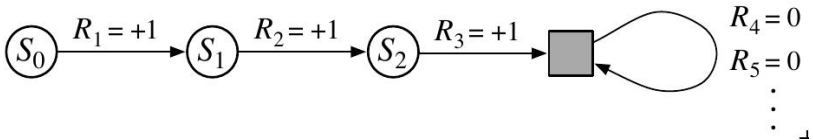
- Can make all episodes infinite by adding a self-loop with reward 0



- Allow $\gamma = 1$ only if sum converges

Long term rewards

- Can make all episodes infinite by adding a self-loop with reward 0

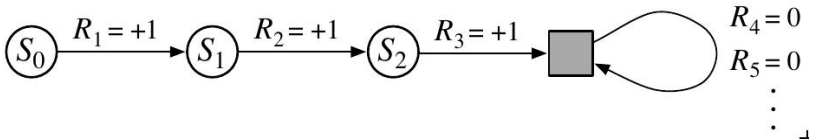


- Allow $\gamma = 1$ only if sum converges
- Alternatively, $G_t \triangleq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$,

  where we allow $T = \infty$ and $\gamma = 1$, but not both at the same time

Long term rewards

- Can make all episodes infinite by adding a self-loop with reward 0



- Allow $\gamma = 1$ only if sum converges
- Alternatively, $G_t \triangleq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$,

  where we allow $T = \infty$ and $\gamma = 1$, but not both at the same time
- If $T = \infty$, $R_k = +1$ for each k, $\gamma < 1$, then $G_t = \frac{1}{1-\gamma}$

Policies and value functions

- A policy π describes how the agent chooses actions at a state
  - $\pi(a \mid s)$ — probability of choosing a in state s, $\sum_a \pi(a \mid s) = 1$

Policies and value functions

- A policy π describes how the agent chooses actions at a state
  - $\pi(a \mid s)$ — probability of choosing a in state s, $\sum_a \pi(a \mid s) = 1$
- State value function at s, following policy π

$$v_\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]$$

Policies and value functions

- A policy π describes how the agent chooses actions at a state
  - $\pi(a \mid s)$ — probability of choosing a in state s, $\sum_a \pi(a \mid s) = 1$
- State value function at s, following policy π

$$v_\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

- Action value function on choosing a at s and then following policy π

$$q_\pi(s,a) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

  - Note that $v_\pi(s) = \sum_a \pi(a \mid s) q_\pi(s,a)$

Policies and value functions

- A policy π describes how the agent chooses actions at a state
  - $\pi(a \mid s)$ — probability of choosing a in state s, $\sum_a \pi(a \mid s) = 1$
- State value function at s, following policy π

$$v_\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

- Action value function on choosing a at s and then following policy π

$$q_\pi(s, a) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

  - Note that $v_\pi(s) = \sum_a \pi(a \mid s) q_\pi(s, a)$
- Goal is to find an optimal policy, that maximizes state/action value at every state

Bellman equation

- $v_\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s]$

Bellman equation

- $v_\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s]$
  $= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$

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

Bellman equation

- $$v_\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s]$$
$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$
$$= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \right]$$

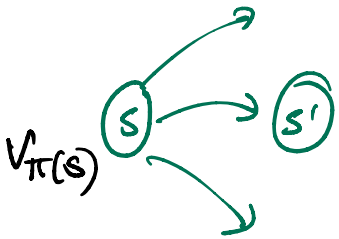
Bellman equation

- $v_\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s]$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \right]$$

$$= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$



Bellman equation

- $v_\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s]$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

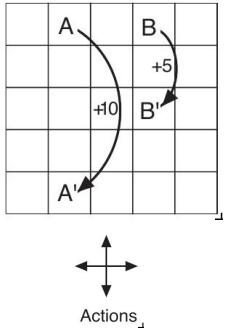
$$= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \right]$$

$$= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$

- Bellman equation relates state value at s to state values at successors of s
- Value function v_π is unique solution to the equation

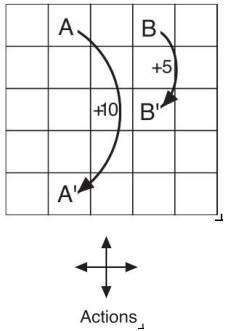
Gridworld Example

- Actions in each cell are {N,S,E,W}, with usual interpretation



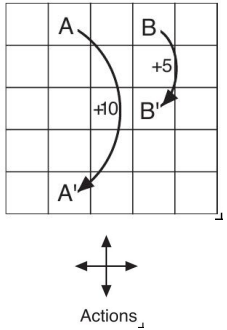
Gridworld Example

- Actions in each cell are {N,S,E,W}, with usual interpretation
- Reward is 0, except at boundaries
- Colliding with boundary — position unchanged, reward -1



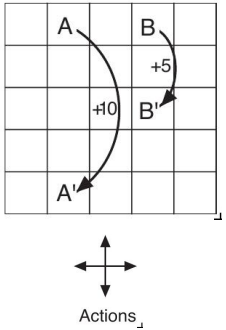
Gridworld Example

- Actions in each cell are {N,S,E,W}, with usual interpretation
- Reward is 0, except at boundaries
- Colliding with boundary — position unchanged, reward -1
- Special squares A and B — all four actions move as indicated, with rewards $+10$ and $+5$, respectively



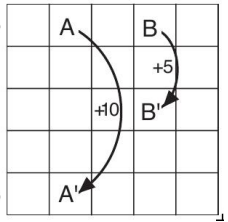
Gridworld Example

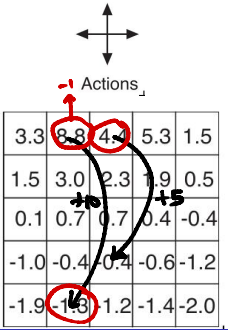
- Actions in each cell are {N,S,E,W}, with usual interpretation
- Reward is 0, except at boundaries
- Colliding with boundary — position unchanged, reward -1
- Special squares A and B — all four actions move as indicated, with rewards $+10$ and $+5$, respectively
- Policy π — choose each action with uniform probability 0.25



Gridworld Example

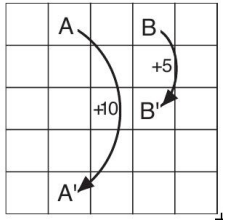
- Actions in each cell are {N,S,E,W}, with usual interpretation
- Reward is 0, except at boundaries
- Colliding with boundary — position unchanged, reward -1
- Special squares A and B — all four actions move as indicated, with rewards $+10$ and $+5$, respectively
- Policy π — choose each action with uniform probability 0.25
- Solving Bellman equations, we obtain v_π for each square





Gridworld Example

- Actions in each cell are {N,S,E,W}, with usual interpretation
- Reward is 0, except at boundaries
- Colliding with boundary — position unchanged, reward -1
- Special squares A and B — all four actions move as indicated, with rewards $+10$ and $+5$, respectively
- Policy π — choose each action with uniform probability 0.25
- Solving Bellman equations, we obtain v_π for each square
- Values at boundary are negative

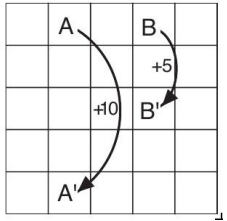


Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Gridworld Example

- Actions in each cell are {N,S,E,W}, with usual interpretation
- Reward is 0, except at boundaries
- Colliding with boundary — position unchanged, reward -1
- Special squares A and B — all four actions move as indicated, with rewards $+10$ and $+5$, respectively
- Policy π — choose each action with uniform probability 0.25
- Solving Bellman equations, we obtain v_π for each square
- Values at boundary are negative
- Value at A is less than 10 because next move takes agent to boundary square with negative value

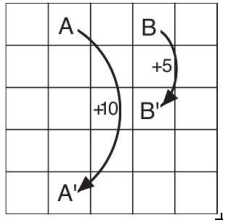


Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Gridworld Example

- Actions in each cell are {N,S,E,W}, with usual interpretation
- Reward is 0, except at boundaries
- Colliding with boundary — position unchanged, reward -1
- Special squares A and B — all four actions move as indicated, with rewards $+10$ and $+5$, respectively
- Policy π — choose each action with uniform probability 0.25
- Solving Bellman equations, we obtain v_π for each square
- Values at boundary are negative
- Value at A is less than 10 because next move takes agent to boundary square with negative value
- Value at B is more than 5 because next move is to a square with positive value



3.9	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Optimal policies and value functions

- Compare policies π, π': $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for every state s

Optimal policies and value functions

- Compare policies π, π': $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for every state s
- Optimal policy π_*, $\pi_* \geq \pi$ for every π
  - Always exists, but may not be unique

Optimal policies and value functions

- Compare policies π, π': $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for every state s
- Optimal policy π_*, $\pi_* \geq \pi$ for every π
  - Always exists, but may not be unique
- Optimal state value function, $v_*(s) \triangleq \max_\pi v_\pi(s) = v_{\pi_*}(s)$

Optimal policies and value functions

- Compare policies π, π': $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for every state s
- Optimal policy π_*, $\pi_* \geq \pi$ for every π
  - Always exists, but may not be unique
- Optimal state value function, $v_*(s) \triangleq \max_\pi v_\pi(s) = v_{\pi_*}(s)$
- Optimal action value function, $q_*(s,a) \triangleq \max_\pi q_\pi(s,a) = q_{\pi_*}(s,a)$

Optimal policies and value functions

- Compare policies π, π': $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for every state s
- Optimal policy π_*, $\pi_* \geq \pi$ for every π
  - Always exists, but may not be unique
- Optimal state value function, $v_*(s) \triangleq \max_\pi v_\pi(s) = v_{\pi_*}(s)$
- Optimal action value function, $q_*(s,a) \triangleq \max_\pi q_\pi(s,a) = q_{\pi_*}(s,a)$
- Bellman optimality equation for v_*

  $v_*(s) = \max_a q_{\pi_*}(s,a)$

Optimal policies and value functions

- Compare policies π, π': $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for every state s
- Optimal policy π_*, $\pi_* \geq \pi$ for every π
  - Always exists, but may not be unique
- Optimal state value function, $v_*(s) \triangleq \max_\pi v_\pi(s) = v_{\pi_*}(s)$
- Optimal action value function, $q_*(s,a) \triangleq \max_\pi q_\pi(s,a) = q_{\pi_*}(s,a)$
- Bellman optimality equation for v_*

$$\begin{aligned} v_*(s) &= \max_a q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \end{aligned}$$

Optimal policies and value functions

- Compare policies π, π': $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for every state s
- Optimal policy π_*, $\pi_* \geq \pi$ for every π
  - Always exists, but may not be unique
- Optimal state value function, $v_*(s) \triangleq \max_\pi v_\pi(s) = v_{\pi_*}(s)$
- Optimal action value function, $q_*(s,a) \triangleq \max_\pi q_\pi(s,a) = q_{\pi_*}(s,a)$
- Bellman optimality equation for v_*

$$
\begin{aligned}
v_*(s) &= \max_a q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a]
\end{aligned}
$$

Optimal policies and value functions

- Compare policies π, π': $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for every state s
- Optimal policy π_*, $\pi_* \geq \pi$ for every π
  - Always exists, but may not be unique
- Optimal state value function, $v_*(s) \triangleq \max_\pi v_\pi(s) = v_{\pi_*}(s)$
- Optimal action value function, $q_*(s,a) \triangleq \max_\pi q_\pi(s,a) = q_{\pi_*}(s,a)$
- Bellman optimality equation for v_*

$$
\begin{aligned}
v_*(s) &= \max_a q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]
\end{aligned}
$$

Optimal policies and value functions

- Compare policies π, π': $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for every state s
- Optimal policy π_*, $\pi_* \geq \pi$ for every π
  - Always exists, but may not be unique
- Optimal state value function, $v_*(s) \triangleq \max_\pi v_\pi(s) = v_{\pi_*}(s)$
- Optimal action value function, $q_*(s,a) \triangleq \max_\pi q_\pi(s,a) = q_{\pi_*}(s,a)$
- Bellman optimality equation for v_*

$$
\begin{aligned}
v_*(s) &= \max_a q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]
\end{aligned}
$$

Bellman optimality equations

- $v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$
  $= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]$

Bellman optimality equations

- $v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$
  $= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]$
- Likewise, for action value function

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = t, A_t = a]$$
$$= \sum_{s',r} p(s', r \mid s, a)[r + \max_{a'} \gamma q_*(s', a')]$$

Bellman optimality equations

- $v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$
  $= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]$
- Likewise, for action value function

  $q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = t, A_t = a]$
  $= \sum_{s',r} p(s', r \mid s, a)[r + \max_{a'} \gamma q_*(s', a')]$
- For finite state MDPs, can solve explicitly for v_*
  - n states, n equations in n unknowns, (assuming we know p)

Bellman optimality equations

- $v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$
  $= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]$
- Likewise, for action value function

  $q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = t, A_t = a]$
  $= \sum_{s',r} p(s', r \mid s, a)[r + \max_{a'} \gamma q_*(s', a')]$
- For finite state MDPs, can solve explicitly for v_*
  - n states, n equations in n unknowns, (assuming we know p)
- However, n is usually large, computationally infeasible
  - State space of a game like chess or Go

Bellman optimality equations

- $v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$
  $= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]$
- Likewise, for action value function

  $q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = t, A_t = a]$
  $= \sum_{s',r} p(s', r \mid s, a)[r + \max_{a'} \gamma q_*(s', a')]$
- For finite state MDPs, can solve explicitly for v_*
  - n states, n equations in n unknowns, (assuming we know p)
- However, n is usually large, computationally infeasible
  - State space of a game like chess or Go
- Instead, we will explore iterative methods to approximate v_*

Policy evaluation

- Given a policy π, compute its state value function v_π

Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n

Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n
- Instead, use the Bellman equations as update rules

Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n
- Instead, use the Bellman equations as update rules
  - Initialize $v_\pi^0(s)$: set $v_\pi^0(\text{term}) = 0$ for terminal state term, arbitrary values for other s

Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n
- Instead, use the Bellman equations as update rules
  - Initialize $v_\pi^0(s)$: set $v_\pi^0(\text{term}) = 0$ for terminal state term, arbitrary values for other s
  - Update v_π^k to v_π^{k+1} using: $v_\pi^{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi^k(s') \right]$

Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n
- Instead, use the Bellman equations as update rules
  - Initialize $v_\pi^0(s)$: set $v_\pi^0(\text{term}) = 0$ for terminal state term, arbitrary values for other s
  - Update v_π^k to v_π^{k+1} using: $v_\pi^{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi^k(s') \right]$
  - Stop when incremental change $\Delta = |v_\pi^{k+1} - v_\pi^k|$ is below threshold θ

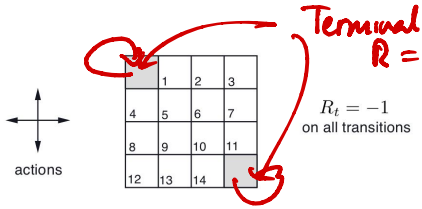
Policy evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

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

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

Policy evaluation example



v_k for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy improvement

- Assume a deterministic policy π
- Using v_π, can we find a better policy π'?

Policy improvement

- Assume a deterministic policy π
- Using v_π, can we find a better policy π'?
- Is there a state s where we can substitute $\pi(s)$ by a better choice a?

Policy improvement

- Assume a deterministic policy π
- Using v_π, can we find a better policy π'?
- Is there a state s where we can substitute $\pi(s)$ by a better choice a?
- $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$

$$= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$

Policy improvement

- Assume a deterministic policy π
- Using v_π, can we find a better policy π'?
- Is there a state s where we can substitute $\pi(s)$ by a better choice a?
- $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$

$$= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$

- If $q_\pi(s, a) > v_\pi(s)$, modify π so that $\pi(s) = a$

Policy improvement

- Assume a deterministic policy π
- Using v_π, can we find a better policy π'?
- Is there a state s where we can substitute $\pi(s)$ by a better choice a?
- $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$
$$= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$
- If $q_\pi(s,a) > v_\pi(s)$, modify π so that $\pi(s) = a$
- The new policy π' is strictly better

Policy improvement

Policy Improvement Theorem

For deterministic policies π, π':

- If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all s, then $\pi' \geq \pi$,
- If $\pi' \geq \pi$ and $q_\pi(s, \pi'(s)) > v_\pi(s)$ for some s, then $v_{\pi'}(s) > v_\pi(s)$.

Policy improvement

Policy Improvement Theorem

For deterministic policies π, π':

- If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all s, then $\pi' \geq \pi$,
- If $\pi' \geq \pi$ and $q_\pi(s, \pi'(s)) > v_\pi(s)$ for some s, then $v_{\pi'}(s) > v_\pi(s)$.

- Proof of the theorem is not difficult for deterministic policies

Policy improvement

Policy Improvement Theorem

For deterministic policies π, π':

- If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all s, then $\pi' \geq \pi$,
- If $\pi' \geq \pi$ and $q_\pi(s, \pi'(s)) > v_\pi(s)$ for some s, then $v_{\pi'}(s) > v_\pi(s)$.

- Proof of the theorem is not difficult for deterministic policies
- The theorem extends to probabilistic policies also

Policy improvement

Policy Improvement Theorem

For deterministic policies π, π':

- If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all s, then $\pi' \geq \pi$,
- If $\pi' \geq \pi$ and $q_\pi(s, \pi'(s)) > v_\pi(s)$ for some s, then $v_{\pi'}(s) > v_\pi(s)$.

- Proof of the theorem is not difficult for deterministic policies
- The theorem extends to probabilistic policies also
- Provides a basis to iteratively improve the policy

Policy iteration

- Start with a random policy π_0

Policy iteration

- Start with a random policy π_0
- Use policy evaluation to compute v_{π_0}

Policy iteration

- Start with a random policy π_0
- Use policy evaluation to compute v_{π_0}
- Use policy improvement to construct a better policy π_1

Policy iteration

- Start with a random policy π_0
- Use policy evaluation to compute v_{π_0}
- Use policy improvement to construct a better policy π_1
- Policy iteration: Alternate between policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\text{evaluate}} v_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluate}} v_{\pi_1} \xrightarrow{\text{improve}} \pi_2 \xrightarrow{\text{evaluate}} \cdots$$

Policy iteration

- Start with a random policy π_0
- Use policy evaluation to compute v_{π_0}
- Use policy improvement to construct a better policy π_1
- Policy iteration: Alternate between policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\text{evaluate}} v_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluate}} v_{\pi_1} \xrightarrow{\text{improve}} \pi_2 \xrightarrow{\text{evaluate}} \cdots \xrightarrow{\text{improve}} \pi_* \xrightarrow{\text{evaluate}} v_{\pi_*}$$

- Finite MDPs — can improve π only finitely many times,
  - Must converge to optimal policy

Policy iteration

- Start with a random policy π_0
- Use policy evaluation to compute v_{π_0}
- Use policy improvement to construct a better policy π_1
- Policy iteration: Alternate between policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\text{evaluate}} v_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluate}} v_{\pi_1} \xrightarrow{\text{improve}} \pi_2 \xrightarrow{\text{evaluate}} \cdots \xrightarrow{\text{improve}} \pi_* \xrightarrow{\text{evaluate}} v_{\pi_*}$$

- Finite MDPs — can improve π only finitely many times,
  - Must converge to optimal policy
- Nested iteration — each policy evaluation is itself an iteration
  - Speed up by using v_{π_i} as initial state to compute $v_{\pi_{i+1}}$

Policy iteration

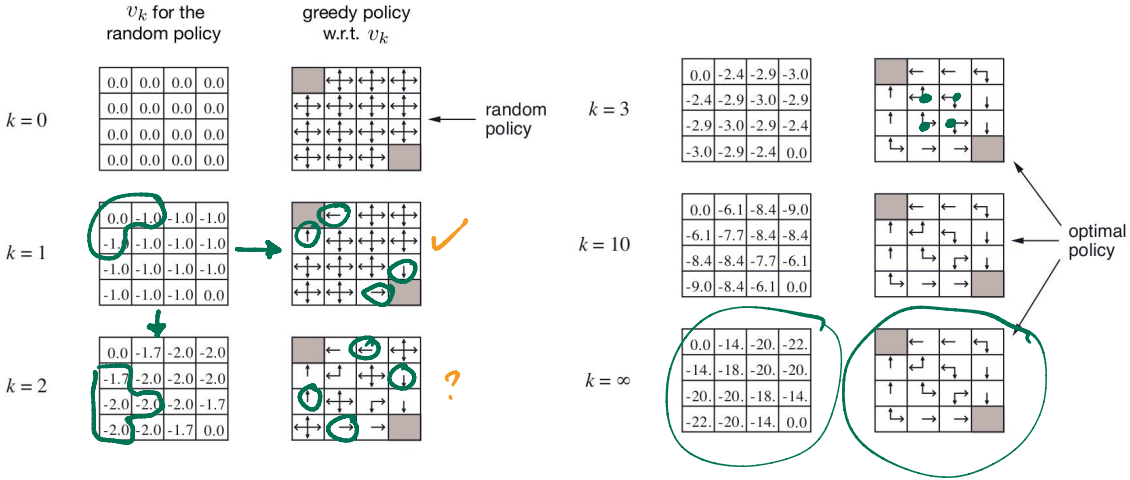
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

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 $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
   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

Optimizing Policy Iteration

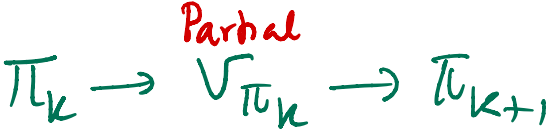


Value iteration

- Policy iteration — policy evaluation requires a nested iteration

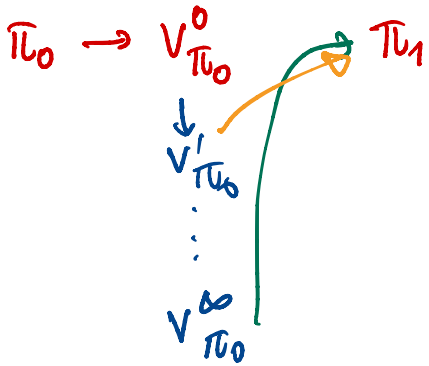
Value iteration

- Policy iteration — policy evaluation requires a nested iteration
- A partial computation of v_{π_k} is sufficent to proceed towards π_*, v_*



Value iteration

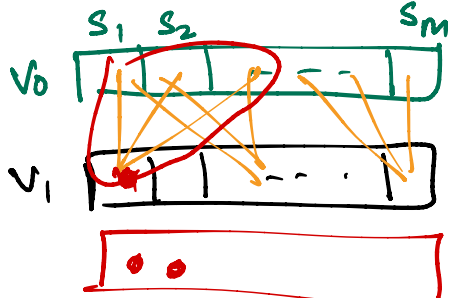
- Policy iteration — policy evaluation requires a nested iteration
- A partial computation of v_{π_k} is sufficent to proceed towards π_*, v_*
- Even a single iteration in the computation of v_{π_k} will do



Value iteration

- Policy iteration — policy evaluation requires a nested iteration
- A partial computation of v_{π_k} is sufficent to proceed towards π_*, v_*
- Even a single iteration in the computation of v_{π_k} will do
- Combine policy improvement and one step update at each state

Even $\quad V_\pi^0 \rightarrow U_\pi'$



Value iteration

- Policy iteration — policy evaluation requires a nested iteration
- A partial computation of v_{π_k} is sufficent to proceed towards π_*, v_*
- Even a single iteration in the computation of v_{π_k} will do
- Combine policy improvement and one step update at each state
- Value iteration

$$v_{\pi_{k+1}}(s,a) = \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi_k}(S_{t+1}) \mid S_t = s, A_t = a]$$

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

Value iteration

- Policy iteration — policy evaluation requires a nested iteration
- A partial computation of v_{π_k} is sufficent to proceed towards π_*, v_*
- Even a single iteration in the computation of v_{π_k} will do
- Combine policy improvement and one step update at each state
- Value iteration

$$v_{\pi_{k+1}}(s,a) = \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi_k}(S_{t+1}) \mid S_t = s, A_t = a]$$

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

- Again, stop when incremental change $\Delta = |v_{\pi_{k+1}} - v_{\pi_k}|$ is below threshold θ

Dynamic programming

- In the literature, policy iteration and value iteration are referred to as dynamic programming methods

Dynamic programming

- In the literature, policy iteration and value iteration are referred to as dynamic programming methods
- Requires knowledge of the model — $p(s', r \mid s, a)$

Dynamic programming

- In the literature, policy iteration and value iteration are referred to as dynamic programming methods
- Requires knowledge of the model — $p(s', r \mid s, a)$
- How to combine policy evaluation and policy improvement is flexible
  - Value iteration is policy iteration with policy evaluation truncated to a single step
  - Generalized policy iteration — simultaneously maintain and update approximations of π_* and v_*

Dynamic programming

- In the literature, policy iteration and value iteration are referred to as dynamic programming methods
- Requires knowledge of the model — $p(s', r \mid s, a)$
- How to combine policy evaluation and policy improvement is flexible
  - Value iteration is policy iteration with policy evaluation truncated to a single step
  - Generalized policy iteration — simultaneously maintain and update approximations of π_* and v_*
- Asynchronous dynamic programming for large state spaces

Other approaches

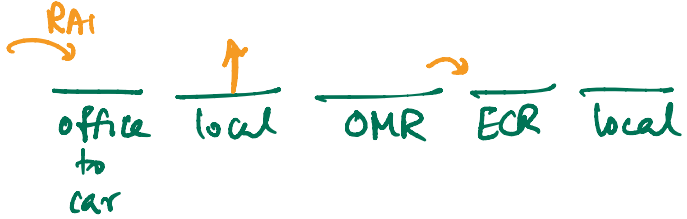
- Monte Carlo methods

Other approaches

- Monte Carlo methods
- Temporal-Difference learning — bootstrap estimates based on observations

Example

- Estimate travel time home = 45 minutes



Go

AlphaGo – used RL to beat Lee Sedol

Raw system

- Teach Go via RL
- Beat AlphaG

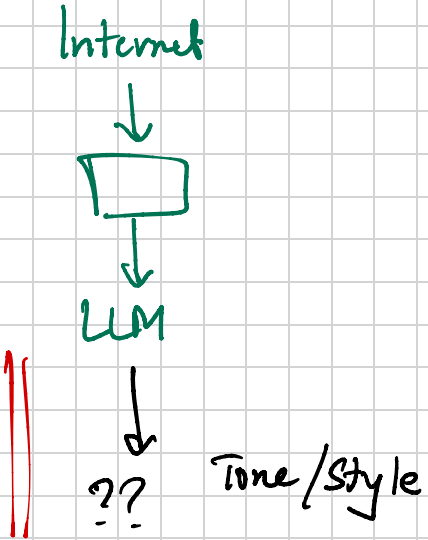
2019 - Yann Le Cun

How many cars will you crash?

LLMs

Pretraining

Fine tuning



Early fine tuning — Supervise

Sample Q & A

Today RLHF — Reinforcement learning with human feedback

Inflection → Pi