Week 2: Mathematical Foundations

Your Name

Your Institution

July 19, 2025

Introduction to Markov Decision Processes (MDPs)

Overview of MDPs

A Markov Decision Process (MDP) is a mathematical framework used to describe an environment in reinforcement learning (RL). It helps in making decisions to achieve goals where outcomes depend on actions and randomness.

Components of MDPs

MDPs are defined by the following key components:

1. **States (S)**: Possible situations or configurations the agent can be in.
2. **Actions (A)**: Choices available to the agent in each state.
3. **Transition Probabilities (P)**:

$$P(s'|s,a) = P(\text{next state is } s' | \text{current state is } s \text{ and action is } a)$$

4. **Rewards (R)**: Numerical values received after transitioning between states.
5. **Discount Factor (γ)**: Value between 0 and 1 determining the present value of future rewards.

Significance of MDPs in Reinforcement Learning

- **Framework for Optimal Decision Making**: MDPs serve as a foundation for developing optimal strategies.
- **Policy Representation**: An agent learns a policy (π) that specifies the actions in each state to maximize cumulative rewards.
- **Value Function**:

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right)$$

Example: Grid World

Consider a simple grid world where:

- **States**: Each grid cell represents a state.
- **Actions**: Up, Down, Left, Right.
- **Transition Probabilities**: Probability of slipping while moving.
- **Rewards**: Positive for reaching specific cells, negative for falling into traps.

The agent explores the grid while learning the best actions to maximize overall reward.

Key Takeaways

- MDPs provide a robust framework for modeling decision-making problems in RL.
- Understanding MDP components aids in formulating and solving complex problems in various fields.
- Mastery of MDPs is crucial for developing effective RL algorithms and policies.

What are MDPs?

Definition of Markov Decision Processes (MDPs)

A Markov Decision Process (MDP) is a mathematical framework used for modeling decision-making situations where outcomes are partly random and partly under the control of a decision-maker.

Significance

MDPs are foundational in areas such as Reinforcement Learning, robotics, and operations research.

Components of MDPs

MDPs consist of four key components:

1. **States (S)**: Represents the current situation of the decision maker in an environment.
2. **Actions (A)**: The set of all possible moves or decisions the decision-maker can make.
3. **Transition Probabilities (P)**: Defines the likelihood of moving from one state to another, given a specific action.
4. **Rewards (R)**: Values received after transitioning between states, representing the immediate benefit of an action.

Detailed Components of MDPs

1. **States (S)**:
   - Example: In a grid world, each cell can be a state, e.g., (2,3).
2. **Actions (A)**:
   - Example: In the grid world, actions may include moving up, down, left, or right.
3. **Transition Probabilities (P)**:
   - Example: $P((2,4)|(2,3), \text{right}) = 0.7$ represents probability of moving from (2,3) to (2,4).
4. **Rewards (R)**:
   - Example: A reward of +10 for reaching a target position, -5 for hitting an obstacle.

Key Points and Mathematical Representation

Key Points to Emphasize:

- **Markov Property**: Future state depends only on the current state and action, not previous history.
- **Goal**: To find a policy that maximizes the expected sum of rewards over time.
- **Real-World Applications**: Used in robotics, finance, and game theory.

Mathematical Representation:

$$MDP = (S, A, P, R) \tag{1}$$

where:

- S = set of states,
- A = set of actions,
- $P : S \times A \times S \to [0, 1]$ = transition probabilities,
- $R : S \times A \to \mathbb{R}$ = reward function.

Wrap-Up

Understanding MDPs will provide the groundwork for exploring more complex models and algorithms in Reinforcement Learning.

Next Steps

The next slide will delve into the mathematical representation and formulation of MDPs, enhancing your understanding of their operational mechanics.

Mathematical Representation of MDPs - Introduction

A Markov Decision Process (MDP) is a mathematical framework for modeling decision-making situations where outcomes are partly random and partly under the control of a decision-maker. It is characterized by a tuple (S, A, P, R, γ):

- **S**: A set of states.
- **A**: A set of actions available to the agent.
- **P**: Transition probabilities, representing the probability of reaching a new state given the current state and action.
- **R**: Reward function detailing the reward received after transitioning from one state to another following an action.
- γ: Discount factor, determining the importance of future rewards compared to immediate rewards.

Mathematical Representation of MDPs - Formulation

1. **States (S)**: Each possible situation in which the agent can find itself. For example:
$$S = \{s_1, s_2, \ldots, s_n\}$$
2. **Actions (A)**: The set of all actions the agent can take in any state. For example:
$$A(s) = \{a_1, a_2, \ldots, a_m\} \text{ for state } s.$$
3. **Transition Function (P)**: Defines how the agent transitions between states, given its current state and action:
$$P(s'|s,a) = \Pr(S_{t+1} = s'|S_t = s, A_t = a)$$
4. **Reward Function (R)**: The immediate reward received after transitioning from state s to state s' due to action a:
$$R(s,a) = \text{Expected reward after taking action } a \text{ in state } s.$$

Mathematical Representation of MDPs - Key Concepts

- **Discount Factor (γ)**: A value between 0 and 1 weighing future rewards relative to present rewards. If $\gamma = 0.9$, future rewards are worth 90% of their value today.

$$V(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, a) V(s')$$

where $V(s)$ represents the value function, indicating the maximum expected future rewards obtainable from state s.

Key Points to Emphasize

- MDPs encourage a structured approach to decision-making.
- Transition probabilities provide a probabilistic approach to uncertainty.
- The discount factor balances immediate versus future rewards.

Dynamic Programming Basics

Introduction to Dynamic Programming

Dynamic Programming (DP) is a method for solving complex problems by breaking them down into simpler subproblems. It is particularly effective for optimization problems and is widely used in the context of Markov Decision Processes (MDPs).

Dynamic Programming - Key Concepts

1. **Optimal Substructure**: The optimal solution to a problem can be constructed from optimal solutions to its subproblems.
2. **Overlapping Subproblems**: DP solves subproblems that are often repeated multiple times. Storing results (memoization) reduces computational time.

Role of Dynamic Programming in MDPs

Markov Decision Processes (MDPs)

MDPs consist of states, actions, transition probabilities, rewards, and policies. Dynamic programming computes the optimal policy through the **Bellman Equation**:

$$V(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right) \tag{2}$$

Where:

- $V(s)$ = Value of state s
- A = Set of all actions
- $R(s,a)$ = Immediate reward for action a in state s
- γ = Discount factor ($0 < \gamma < 1$)
- $P(s'|s,a)$ = Transition probability to the next state s' given state s and action a

Example: Fibonacci Sequence

Dynamic Programming Example

A classic example of DP is computing Fibonacci numbers:

- **Naive approach**: $F(n) = F(n-1) + F(n-2) \rightarrow$ Exponential time.
- **DP approach**: Store computed values to avoid recomputation.

```
def fibonacci(n):
    fib = [0, 1] + [0] * (n - 1)
    for i in range(2, n + 1):
        fib[i] = fib[i - 1] + fib[i - 2]
    return fib[n]
```

Conclusion

Conclusion

Dynamic programming is a powerful tool for solving MDPs. By understanding its principles—optimal substructure and overlapping subproblems—students can efficiently tackle a variety of decision-making problems in uncertain environments.

Value Iteration Algorithm - Introduction

- Value Iteration is an algorithm for computing optimal policies in Markov Decision Processes (MDPs).
- Systematically updates value estimates for each state until convergence.

Key Concepts

- **Markov Decision Process (MDP)**: Framework for decision-making with random outcomes.
- **State (S)**: Current situation of the process.
- **Action (A)**: Choices available to the decision-maker.
- **Value Function (V)**: Maximum expected utility from each state.
- **Discount Factor (γ)**: Value between 0 and 1 that discounts future rewards.

Value Iteration Algorithm: Steps

1. **Initialization:**
   - Start with arbitrary state values (often zero).
   - Set convergence threshold ϵ.
2. **Update Values:**
   - For each state $s \in S$:

$$Q(s,a) = \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V(s')\right] \tag{3}$$

   - Update using:

$$V_{\text{new}}(s) = \max_a Q(s,a) \tag{4}$$

3. **Check for Convergence:**
   - Calculate maximum change:

$$\Delta = \max_{s \in S} |V_{\text{new}}(s) - V(s)| \tag{5}$$

   - If $\Delta < \epsilon$, stop. Otherwise, update and repeat.
4. **Extract Optimal Policy:**

$$\pi^*(s) = \arg\max_a Q(s,a) \tag{6}$$

Example Illustration

- Consider an MDP with 2 states s_1 and s_2.
- Transition probabilities and rewards defined.
- Initialize values: $V(s_1) = 0$, $V(s_2) = 0$.
- Update values based on actions and compute until convergence.

Key Points to Emphasize

- Value Iteration guarantees finding the optimal value function for a small enough threshold ϵ.
- Convergence typically occurs in a few iterations due to the nature of MDPs.
- It is computationally efficient as it does not require holding all policy data.

Conclusion

- Understanding Value Iteration sets the foundation for exploring policies.
- It leads to effective decision-making strategies under uncertainty, paving the way for advanced topics such as Policy Iteration.

Policy Iteration Algorithm - Overview

Overview

The Policy Iteration algorithm is a fundamental method used in Reinforcement Learning (RL) and Decision Making to find the optimal policy in Markov Decision Processes (MDPs). An optimal policy defines the best action to take in each state to maximize cumulative rewards.

- **Policy**: A strategy specifying actions for each state.
- **Value function**: Estimation of expected return from each state under a policy.
- **Optimal Policy**: Achieves the highest expected reward over time.

Policy Iteration Algorithm - Workflow

1. **Initialization**: Start with an arbitrary policy π and value function $V(s)$.
2. **Policy Evaluation**: Calculate $V(s)$ for policy π:
$$V(s) = R(s) + \gamma \sum_{s'} P(s'|s,a)V(s') \tag{7}$$
3. **Policy Improvement**: Update to a new policy π':
$$\pi'(s) = \arg\max_a \left(R(s) + \gamma \sum_{s'} P(s'|s,a)V(s') \right) \tag{8}$$
4. **Convergence Check**: If $\pi' = \pi$, converge; otherwise, update π and repeat.

Policy Iteration Algorithm - Example and Key Points

Example

Consider a simple grid world MDP:

- **States**: Each cell represents a state.
- **Actions**: Movements (Up, Down, Left, Right).
- **Rewards**: Costs for each step; rewards for reaching the goal.

1. Initialize a random policy (e.g., always go 'Right').
2. Evaluate the policy.
3. Improve based on calculated values.
4. Repeat until stabilization.

Key Points

- Guarantees convergence to optimal policy.
- Consists of two main steps: Evaluation and Improvement.
- May require more resources than other methods (e.g., Value Iteration)

Understanding MDPs

Markov Decision Processes (MDPs) provide a mathematical framework for modeling decision-making in environments where outcomes are partly random and partly under the control of a decision-maker.

- **States (S)**: The various situations in which an agent can find itself.
- **Actions (A)**: The set of all possible actions that the agent can take.
- **Transition Function (P)**: A probability distribution that describes the likelihood of moving from one state to another given an action.
- **Reward Function (R)**: A function that measures the immediate reward received after transitioning from one state to another via a specific action.
- **Discount Factor (γ)**: A factor between 0 and 1 that balances immediate and future rewards.

Real-World Applications of MDPs

1. Gaming

In strategy games like Chess or Go, an AI uses MDPs to assess the best moves by evaluating the current state of the board and predicting future states.

- Players can think of the current board configuration as a state.
- Actions lead to new board configurations with associated rewards (winning the game).

2. Robotics

Autonomous robots, such as drones or self-driving cars, utilize MDPs for navigation and decision-making.

- The robot's current location is a state.
- Actions include turning or accelerating.
- Rewards can be based on reaching a destination quickly while avoiding collisions.

Continued Applications of MDPs

3. Finance

MDPs model problems in the finance sector, e.g., portfolio management.

- States represent the current portfolio configuration.
- Actions involve buying or selling stocks.
- Rewards are the profits generated from these actions.

Key Points to Emphasize

- **Decision-Making Framework**: MDPs provide structured methods to evaluate choices.
- **Handling Uncertainty**: MDPs are robust in environments with randomness, making them suitable for real-world applications.
- **Adaptability**: The framework can be adapted to a variety of domains.

Conclusion and Next Steps

MDPs are a powerful tool in reinforcement learning, enabling the development of intelligent systems across various fields. They model complex decision-making scenarios crucial for technologies like gaming AI, robotics, and financial forecasting.
Next Steps: In the following slide, we will discuss the challenges and limitations encountered when applying MDPs, including state space complexity and computational demands.

Challenges in MDPs - Overview

- MDPs model decision-making in uncertain environments.
- Several challenges exist when implementing MDPs effectively:
    1. State Space Complexity
    2. Computational Complexity
    3. Partial Observability
    4. Scalability

Challenges in MDPs - State Space Complexity

Definition

The state space is the set of all possible situations an agent can encounter, crucial for MDP design.

- **Challenge**: The curse of dimensionality leads to enormous state spaces, complicating storage and computation.
- **Example**: In robotic navigation, each position and orientation in a grid results in a rapidly increasing state space size.

Challenges in MDPs - Computational Complexity

Definition

Refers to the time and space resources needed to solve MDPs.

- **Challenge**: Algorithms like Value Iteration and Policy Iteration are computationally intensive.
- **Important Note**: Time to compute an optimal policy can increase exponentially with state space size, limiting real-time application.

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V_k(s')\right] \tag{9}$$

Where $P(s'|s,a)$ is the state transition probability, R is the reward function, γ is the discount factor, and V is the value function.

Challenges in MDPs - Other Considerations

- **Partial Observability**: Agents may lack complete information about current states, leading to Partially Observable MDPs (POMDPs) which are harder to solve.
- **Scalability**: As MDPs grow, effectively scaling them to real-world applications can require reducing state space via approximations or abstractions.

Examples of State Space Complexity

- **Robotics**: A robotic arm's state includes its position, object locations, and gripper state. Complex environments lead to unmanageable state spaces.
- **Gaming**: Video games generate immense states from character actions; considering all combinations over time results in significant computational demands.

Key Points and Conclusion

- Challenges of state space and computational complexity highlight the need for approximations or alternative methods in MDPs.
- Understanding these challenges is essential for designing effective algorithms for practical MDP applications.

Conclusion

MDPs are powerful decision-making frameworks, but the challenges associated with state space complexity and computational demands require innovative approaches.

Ethics and Implications of MDPs

Introduction

Markov Decision Processes (MDPs) are foundational in Reinforcement Learning (RL) for decision-making in uncertain environments. Ethical considerations become crucial as MDPs are applied across various domains.

Key Ethical Considerations

1. **Autonomy**
    - Definition: The right of individuals to make their own choices.
    - Implication: Must respect human autonomy in automated decisions.
2. **Fairness**
    - Definition: Treating all individuals or groups equally and without bias.
    - Implication: MDPs can reinforce biases present in training data.
3. **Accountability**
    - Definition: Obligation to explain decisions.
    - Implication: Complex responsibility in adverse outcomes from MDPs.
4. **Privacy**
    - Definition: Control over personal information.
    - Implication: Responsible handling of data collected by MDPs is crucial.

Potential Impacts of MDPs

1. **Societal Transformation**
    - MDPs can enhance efficiency in various sectors.
2. **Decision-making Quality**
    - High-quality decision-making possible but poor implementation can lead to disasters.
3. **Job Displacement**
    - Automating tasks may lead to job losses, requiring workforce redefinition.
4. **Environmental Effects**
    - Potential environmental benefits; mismanagement could worsen issues.

Example Illustration

Example Scenario

A hospital using an MDP-based system to allocate emergency resources:

- Prioritizing patients solely on data-driven metrics, without individual consideration, can lead to unfair treatment.
- Ethical decision-making requires a balance of algorithms with human oversight.

Conclusion and Key Takeaways

Conclusion

The application of MDPs holds immense potential but requires careful consideration of ethical implications.

- MDPs must be applied with respect to ethics.
- Awareness of autonomy, fairness, accountability, and privacy is paramount.
- Societal impacts of MDP applications must be managed responsibly.

Important Quote

“With great power comes great responsibility.”

Summary and Key Takeaways - Overview

- Recap of MDPs as a framework for decision-making.
- Dynamic programming techniques for reinforcement learning.
- Essential insights and real-world applications.

Recap of Markov Decision Processes (MDPs)

1. **Definition of MDPs**:
   - **States (S)**: All possible states the agent can occupy.
   - **Actions (A)**: All possible actions the agent can take.
   - **Transition Function (T)**: $T(s, a, s') = P(s'|s, a)$.
   - **Reward Function (R)**: Rewards received: $R(s, a)$.
   - **Discount Factor (γ)**: Between 0 and 1, for future reward importance.
2. **Key Properties**:
   - **Markov Property**: Next state depends only on the current state and action.
   - **Policy (π)**: Strategy for action selection in each state.

Dynamic Programming in Reinforcement Learning

1. **Overview**:
   - Algorithms that solve MDPs by simplifying problems.
   - Essential for finding optimal policies in reinforcement learning.
2. **Key Methods**:
   - **Value Iteration**:

$$V_{k+1}(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_k(s') \tag{10}$$

   - **Policy Iteration**: Policy evaluation and improvement.
   - **Example**: Robot navigating a grid world with states, actions, and rewards.

Key Insights and Applications

- **Applications**: Used in operations research, robotics, and economics.
- **Challenges**: Large state/action spaces lead to computational difficulties.
- **Approaches**: Development of approximations and simulation-based methods in reinforcement learning.

Key Takeaways

- MDPs offer a structured approach to dynamic decision problems.
- Dynamic programming is crucial for optimizing decision policies.
- Understanding MDPs helps analyze ethical implications in their applications.

Formula Recap

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s') \right) \tag{11}$$